

Глава 1. Отладка и тестирование приложений

Отладка — это процесс пошаговой проверки программ и приложений с целью выявления ошибок.

Процесс отладки характеризуется тем, что программа останавливается каждый раз в точках прерывания. Если вы забыли установить точки прерывания, то отладка не будет отличаться от обычного запуска.

Рассмотрим отладку на примере. Создайте новый класс с именем `Debuging` и введите предложенный в листинге код.

```
public class Debuging {
    public static void main(String[] args){
        for (int i=0;i<10;i++){
            System.out.print("Шаг "+i+"\n");
        }
    }
}
```

Для запуска отладчика нажмите кнопку с изображением жука.



При нажатии данной кнопки включается режим отладки текущей программы, если окно кода программы активно. Если активна панель `Package Explorer`, то отладке подвергнется выбранный (выделенный) класс, при этом класс можно дополнительно выбрать, если их несколько.

На рис. 21 выделено поле, в котором пользователь может установить точки прерывания.

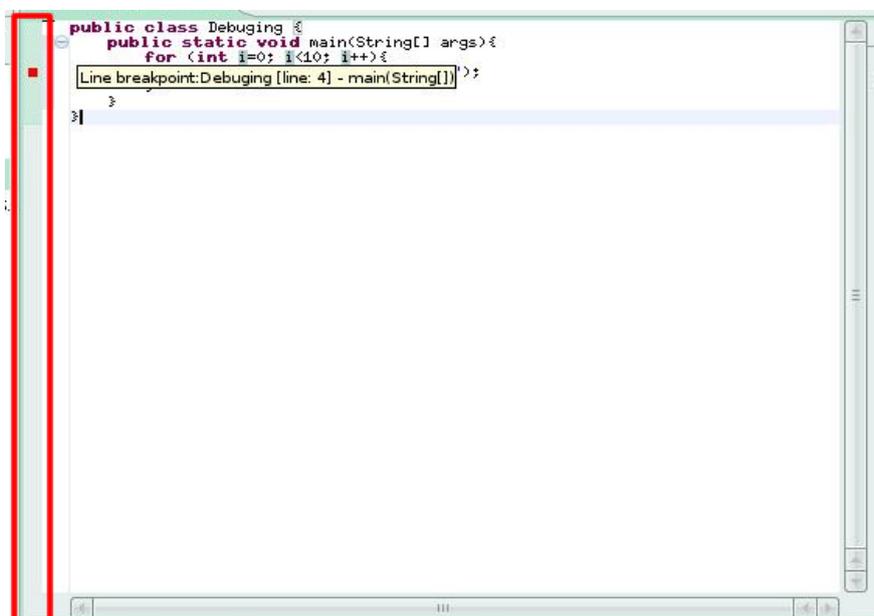


Рис. 21. Линейка точек прерывания

Для того чтобы поставить точку, нужно дважды щелкнуть мышью в данном поле напротив нужной строки, где планируется остановка программы. Маркеры точек прерывания представляют собой небольшие красные квадраты.

Установите точку прерывания напротив строки

```
System.out.print ("Шар "+i+"\n");.
```

После того, как вы установите точку прерывания и нажмете кнопку , на экране появится диалоговое окно (рис. 22), предлагающее переключить режим рабочего стола из Java в Debug.

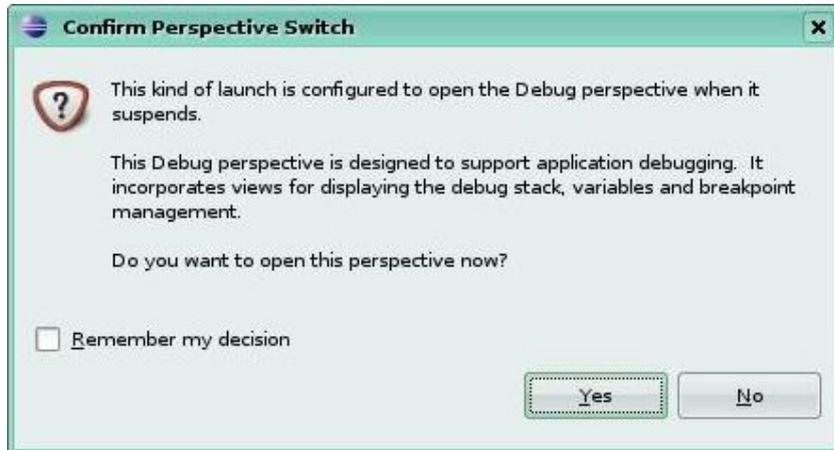


Рис. 22. Предложение переключения рабочего стола

Для того чтобы переключиться обратно в режим Java, используйте кнопку , находящуюся в верхнем правом углу рабочего стола. Далее, примите данное предложение, нажав кнопку «Yes», и в итоге рабочий стол примет вид, представленный на рис. 23.

Это стандартный, по умолчанию, вид компоновки Debug, но вы можете его изменить, удаляя, добавляя или перетаскивая панели мышью.

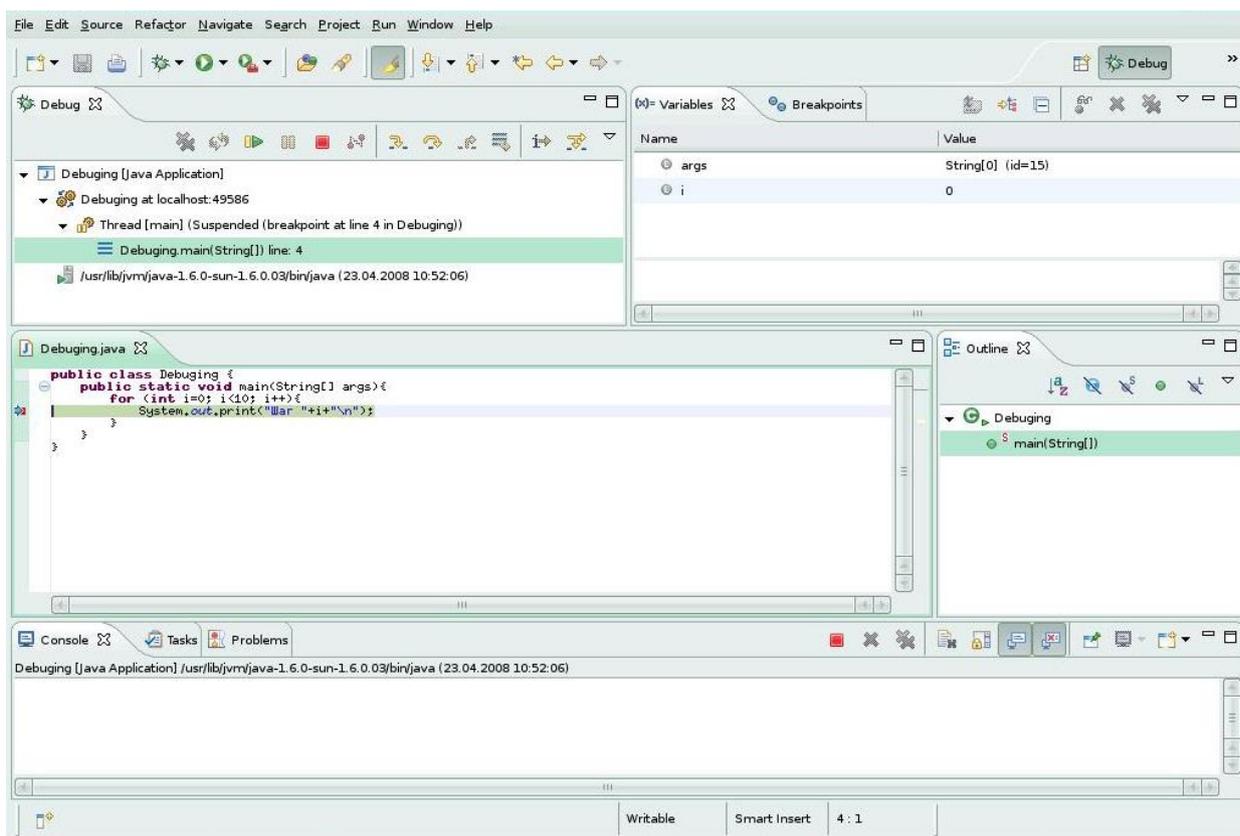


Рис. 23. Рабочий стол Debug

В компоновке Debug можно выделить следующие основные компоненты:

- Окно Debug. В данном окне отображаются задействованные в отладке элементы, а так же панель управления процессом отладки (рис. 24).

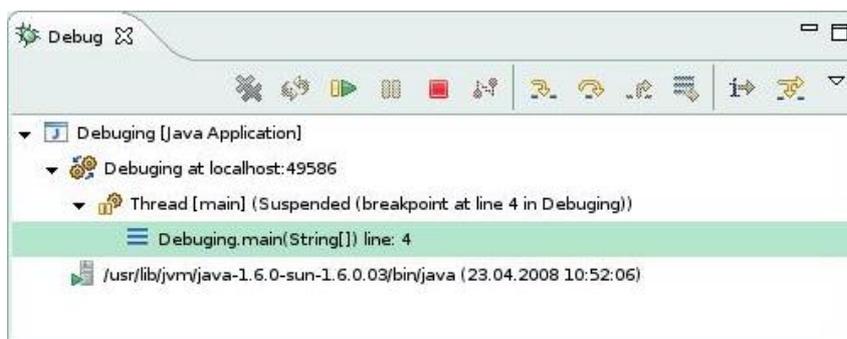


Рис. 24. Окно Debug

- Окно состояния переменных и точек прерывания. Вкладка Variables содержит список переменных, задействованных в текущей точке прерывания. Данное окно является наиболее важным в процессе отладки, оно производит мониторинг переменных на каждой точке прерывания. Именно с помощью данного окна программист может увидеть поведение объекта, изменение его свойств более тщательно в процессе выполнения программы. Вкладка Breakpoints содержит список точек прерывания, которые установил программист. Точки прерывания

можно отключать и включать в процессе отладки, устанавливая или удаляя маркер напротив нужной точки (рис. 25). В левом столбце указаны идентификаторы (имена) переменных, в правом — их текущее значение.

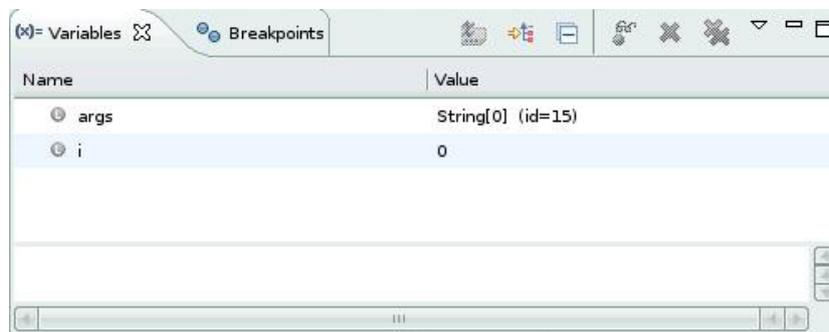


Рис. 25. Окно переменных

- Панель Outline, редактор кода и консоль. Назначение данных элементов рассмотрено ранее.

Теперь программа запущена в режиме отладки. Последовательно выполняя операторы программы, отладчик доходит до первой точки и останавливает ход выполнения. Программист имеет возможность посмотреть значение переменных программы на момент останова. После того как вы проанализируете состояние программы, ее можно продолжать. Нажмите кнопку  («Resume») или клавишу F8 для продолжения.

В нашем случае программа была остановлена на операторе вывода данных на консоль. После продолжения выполнения тело цикла продолжит выполняться, так как в теле нашего цикла только один оператор, он выполнится, и на консоль выведется первая строка, на первом шаге переменная $i=0$, в этом можно убедиться просмотрев вкладку Variables окна состояния (рис. 26).

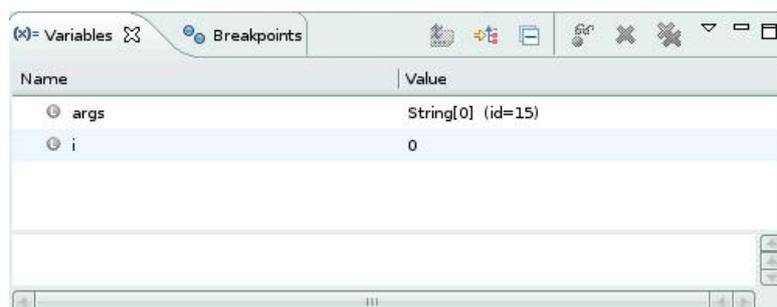


Рис. 26. Состояние переменных на первом шаге

После вывода строки цикл начнет выполняться еще раз, переменная i изменится и станет равной 1. При этом в панели Variables она выделится желтым цветом, что сигнализирует о изменении ее значения (рис. 27).

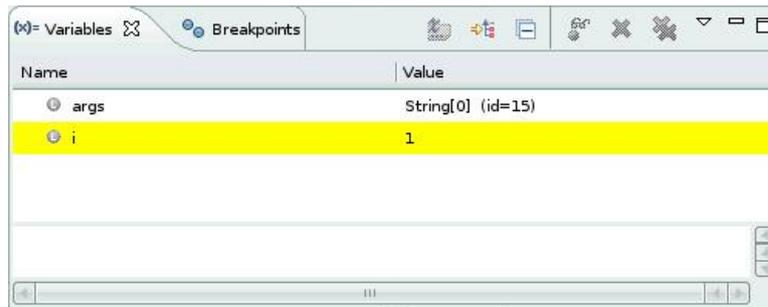


Рис. 27. Состояние переменных на втором шаге

И так далее, нажимая клавишу F8 или кнопку , программа будет переходить от одной точки прерывания к другой. Если точка прерывания установлена на оператор, находящийся в цикле, программа в этом случае будет останавливаться на этой точке столько раз, сколько раз будет проходить цикл.

В отладчике Eclipse есть еще одна полезная функция, с помощью которой можно увидеть значения сложных выражений. Например, нас интересует значение логического выражения (условие), при котором выполняется цикл. Для этого выделите нужный текст, в нашем случае это `i < 10` в блоке условия цикла, и нажмите сочетание клавиш `<Ctrl>+<Shift>+D`, появится всплывающее окно, в котором будет отображен тип выражения (в нашем случае это `boolean`) и его состояние.

Когда программа дойдет до конца, отладка закончится. Для повтора отладки нужно еще раз нажать кнопку .

Для прерывания работы отладчика нажмите кнопку  в панели Debug.

Помимо отладки с использованием безусловных точек прерывания, возможно дополнительно установить условия останова на точке прерывания. Это очень полезная особенность отладчика Eclipse, которая позволяет в некоторых случаях значительно упростить процесс отладки.

Для того, чтобы задать условие, при котором будет происходить останов, нужно открыть окно свойств точки прерывания. Для этого наведите курсор мыши на нужную точку прерывания в строке, нажмите правую кнопку мыши, и выберите в раскрывающемся меню пункт `Breakpoint Properties`, откроется окно свойств выбранной точки прерывания (рис. 28).

Установите галочку на кнопке выбора `Enable Condition` (Включить Условие) и в поле ввода введите необходимое условие, при котором будет происходить останов на данной точке. Для примера, введите логическое выражение `(i % 2) == 0`. Данное выражение будет принимать значение «истина» каждый раз, когда остаток от деления `i` на 2 будет равен 0, то есть каждый раз, когда `i` будет принимать четные значения. Запустите процесс отладки и убедитесь в том, что теперь останов будет происходить каждые два прохода цикла, а не один, как было раньше.

Так же можно выбрать тип условия прерывания — ниже, под полем ввода находится две кнопки выбора — `condition is 'true'` и `value of condition changes`. При выборе первого, останов будет происходить каждый раз, когда введенное логическое выражение

примет значение «истина», при выборе второго, останов будет происходить каждый раз, когда введенное значение или константа изменит свое значение.

Раскрывающийся список Suspend Policy устанавливает политику останова. При выборе значения Suspend Thread останов будет происходить на уровне текущего потока thread, при выборе Suspend VM — на уровне самой виртуальной машины Java.

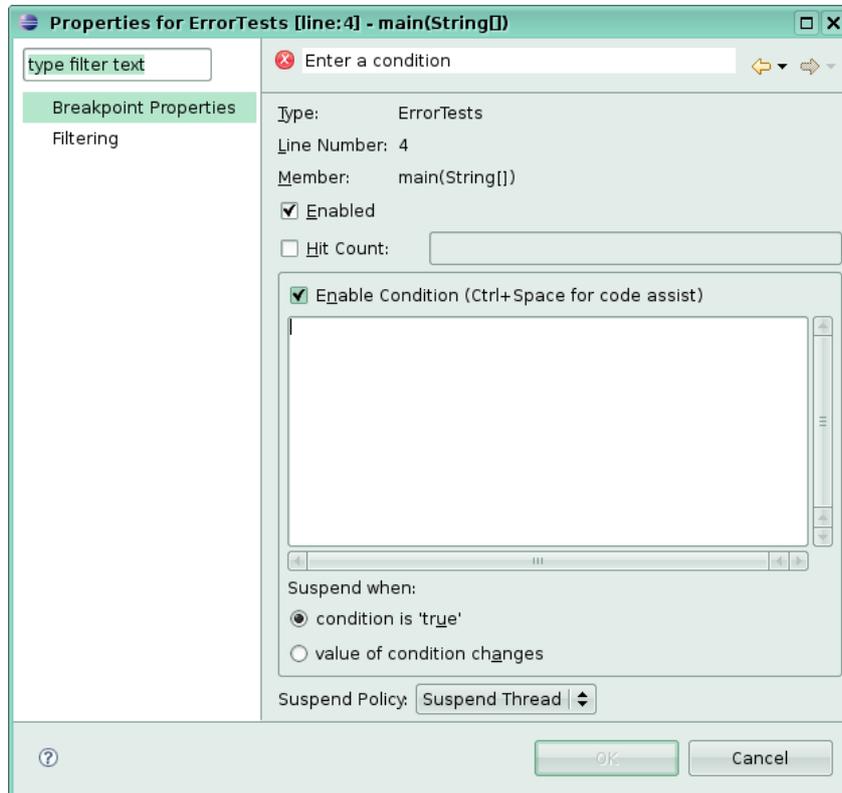


Рис. 28 Свойства точки прерывания

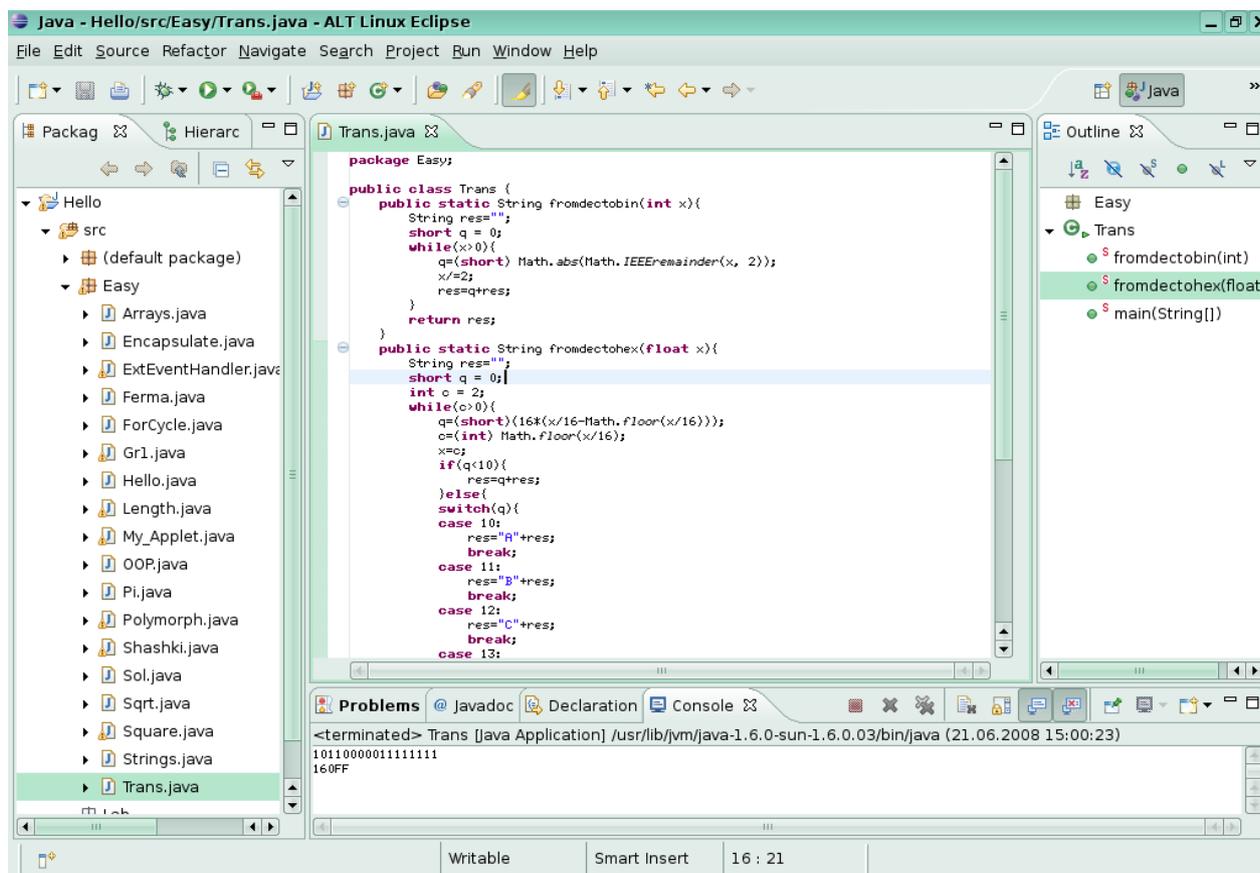
Глава 2. Лабораторный практикум

Лабораторная работа №1

Тема работы. Введение в среду программирования Eclipse.

Цель работы. Начальное знакомство с IDE Eclipse.

Содержание работы. Рассматриваются базовые компоненты интерфейса пользователя Eclipse. Компоненты, представления. Создание проекта, создание и операции с классами.



Задания к работе

1. Установите последовательно компоненты на вашем рабочем столе: Java, Java Browsing, Debug. Укажите функции и назначение этих компонентов.
2. Работа с представлениями. В компоненту Java добавьте новые представления: Problems, Members и затем их закройте. В компоненту Java Browsing добавьте новые представления: Debug, Display, Memory и затем их закройте.
3. Создайте новый проект newProject1. Добавьте в него три пустых класса: wnclass1, wnclass2, wnclass3.
4. Переименуйте классы, созданные в задании 3: fqrs1, fqrs2, fqrs3.

5. Измените место расположение на диске классов поместив их в новый предварительно созданный каталог New.
6. Удалите проект созданный в задании 3.

Лабораторная работа №2

Тема работы. Введение в язык программирования Java.

Цель работы. Первое знакомство с языком программирования Java.

Содержание работы. В данной работе подробно рассматриваются программы, на основе которых учащиеся познакомятся со структурой простейших программ Java.

Программа 1. Простейшая программа.

```
01 | public class Hello {
02 |     public static void main(String[] args) {
03 |         System.out.print("Добро пожаловать в мир Java!");
04 |     }
05 | }
```

Внимание: с целью облегчения комментирования текста программ практикума в листинги программ добавлена нумерация строк, которую вы не найдете в редакторе кода Eclipse.

Данная программа выводит сообщение на консоль (или в командную строку). Приводится описание класса с именем Hello в строке (01), в котором имеется только один метод main() (02). При запуске программы управление передается этому методу и сразу же происходит вывод сообщения на консоль.

В самом начале не нужно пытаться понять, как устроен процесс вывода, он достаточно сложный, нужно просто помнить, что вывод производится конструкцией, называемой *объектом стандартного вывода* System.out с использованием метода print. Метод println позволяет после вывода строки осуществить переход курсора на новую строку.

Программа 2. Вывод текста несколькими строками.

```
01 | public class Hello2 {
02 |     // Выполняется приложение Java начинается с метода main
03 |     public static void main(String[] args) {
04 |         System.out.println("Добро\n пожаловать\n в мир\n Java!");
05 |     } // окончание метода main
06 | } // окончание класса Hello2
```

Управляющие Escape последовательности: \n-новая строка, \t- перемещение на следующую позицию табуляции, \r-в начало текущей строки, \|-обратный слеш, \» - двойные кавычки.

Задания к работе

1. Вывести на консоль следующий набор символов:
a=1, c=a+1 m=2, n=3 решение уравнений
b=6, d=b*2-1 p=4, g=5 нахождение корней
2. Вывести на консоль следующий набор символов:

```
1 2 3
1 2 3 4 5
1 2 3 4 5 6 7
```

Лабораторная работа №3

Тема работы. Алгоритмизация и использование управляющих структур в Java.

Цель работы. Использование циклов в приложении Java.

Содержание работы. Рассмотрены программы, в которых используются операторы циклов For, While для многократного выполнения определенных операторов.

Программа 1. Вычисление суммы и произведения последовательности из 10 случайных чисел.

```
01 public class ForCycle {
02     public static void main(String[] args) {
03         int tmp = 0;
04         long tmp2 = 1;
05         System.out.print("Сумма\t\tПроизведение \n");
06         for(int i=0;i<10;i++){
07             tmp += (int)Math.round(Math.random()*10);
08             tmp2 = tmp2*tmp;
09             System.out.print(tmp+"\t\t"+tmp2+"\n");
10         }
11     }
12 }
```

В строке 02 объявляем главный метод `main()`. В объявлении переменной `tmp` типа `int` присваиваем ей начальное значение (`tmp=0`, инициализация переменной `tmp` в строке 03). В переменной `tmp` будет храниться сумма 10 членов последовательности из случайных чисел. Далее объявляем переменную `tmp2` типа `long`, в которой будем хранить значения произведения членов последовательности (04). В цикле FOR (06-10) находится сумма и произведение элементов последовательности. Генерация случайного члена последовательности осуществляется с использованием стандартной функции `Math.random()`, которая возвращает случайное число в диапазоне от 0 до 1. Далее умножаем случайное число на 10 и округляем `Math.round()`. Печать результатов производится в строке 09.

Программа 2. Вычисление квадратного корня числа с помощью итерационной формулы Герона.

```
01 public class Sqrt {
02     static void sqrt(long a){
03         double b=a;
04         int i=0;
05         while ((b*b>a)&&(i<200)){
06             b=(b+a/b)/2;
07             i++;
08         }
09         System.out.print(b);
10     }
11     public static void main(String[] args) {
```

```

12 |      sqrt(45);
13 |    }
14 | }

```

Для вычисления квадратного корня числа используем итерационную формулу Герона $X_{n+1} = \frac{1}{2}(X_n + \frac{a}{X_n})$. Для этого используем цикл `while` с предусловием `((b*b>a) && (i<200))`.

В строке 02 объявляем новый метод `sqrt()` с одним параметром `a` типа `long`. В объявлении переменной `b` типа `double` присваиваем ей начальное значение (`b=a`, инициализация переменной `b` в строке 03). В переменной `b` будем хранить промежуточные значения корня, вычисляемые в цикле по формуле Герона `b=(b+a/b)/2`. Далее объявляем переменную `i` типа `int` и инициализируем ее значением 0. Данная переменная понадобится в качестве счетчика цикла.

Далее следует цикл `While` (05-08) с предусловием `((b*b>a) && (i<200))`. Данное условие состоит из двух условий: квадрат `b` должен быть больше начального значения `a` и значение счетчика не должно превышать 200, то есть всего производится не более 200 итераций. В данном цикле производится вычисление очередного значения `Xn`, которое хранится в переменной `b` (06). Переменная `b` на каждом шаге цикла изменяется, причем новое значение данной переменной зависит от предыдущего значения. В конце цикла увеличиваем счетчик `i` на 1 в строке 07.

В конце метода выводим полученное значение `b` в консольное окно.

В методе `main()` вызывается метод `sqrt(45)`. В качестве параметра можно взять произвольное целое число (45), которое будет присвоено переменной `a`. (Помните, вызов каких-либо методов напрямую из метода, имеющего модификатор `static`, возможен только при условии, что вызываемый метод тоже является статическим).

Задания к работе

1. С помощью цикла вычислите значение выражения 2^n .
2. Составьте программу расчета факториала для произвольного числа $n < 10$.
3. Даны два действительных числа. Необходимо получить их сумму, разность и произведение. Результат вывести на консоль.
4. Определить время свободного падения материального тела с заданной высоты h . Результат вывести в консоль.
5. Составьте программу для нахождения длины катета прямоугольного треугольника (b), если известны длины гипотенузы (c) и катета (a). Результат вывести на консоль.

Лабораторная работа №4

Тема работы. Алгоритмизация и использование управляющих структур в Java.

Цель работы. Использование элементов организации ветвления в Java.

Содержание работы. В данной работе рассматриваются программы, использующие операторы ветвления. Ветвление используется для организации различных направлений вычислительного процесса.

Программа 1. Вычисления числа Пи.

```
01 public class Pi {
02     static double pi;
04     static void leibnic() {
05         for(double i=1;i<100000000;i+=1){
06             if (i%2==0){
07                 pi-=1/(2*i-1);
09             }else{
10                 pi+=1/(2*i-1);
12             }
13         }
14         pi*=4;
15         System.out.print("\nЧисло Пи, подсчитанное по методу
Лейбница равно: "+pi);
16     }
17     static void vallis(){
18         double pi1=1, pi2=1;
19         for(double i=2;i<100000000;i+=2){
20             pi1*=i/(i+1);
21             pi2*=i/(i-1);
22         }
23         pi=pi1*pi2*2;
24         System.out.print("\nЧисло Пи, подсчитанное по методу Валлиса
равно: "+pi);
25     }
26     public static void main(String[] args) {
27         leibnic();
28         vallis();
29     }
30 }
```

В данной программе описаны два метода нахождения числа Пи: метод Лейбница (ряд Лейбница) и метод Валлиса (формула Валлиса).

В данной программе в строках 02 и 03 описываются `static` переменная класса `pi` типа `double`. Расчеты методом Лейбница вызываются в строке 27 благодаря использованию имени метода `leibnic()`. Аналогично в строке 28 вызываем метод Валлиса `vallis()`.

В методе Лейбница для вычисления числа Пи используется ряд:

$$\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \dots = \frac{\pi}{4}$$

Для вычисления данного ряда мы использовали цикл с условным оператором, где проверяется условие на четность $i\%2==0$, если оно верно, то дробь имеет положительный знак $pi+=1/(2*i-1)$, если нет, то ставим знак минус. В конце, для получения приближенного значения числа Пи, переменная pi умножается на 4 и выводится на консоль.

Расчет по формуле Валлиса

$$\frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \frac{8}{7} \cdot \frac{8}{9} \dots = \frac{\pi}{2}$$

происходит немного по-другому. В методе объявляются две переменные $pi1$ и $pi2$ типа `double` в строке 18, первая хранит произведение четных дробей, вторая — нечетных, затем их удвоенное произведение присваивается переменной pi и выводится на консоль.

Программа 2. Нахождение решения уравнения $A^3 + B^3 + C^3 = \overline{ABC}$

```

01 public class Sol {
02     public static void work() {
03         long x=0;
04         for(int i=1;i<10;i++){
05             for (int j=0;j<10;j++){
06                 for (int k=0;k<10;k++){
07                     x=i*100+j*10+k;
08                     if (x==(Math.pow(i,3)+Math.pow(j, 3) + Math.pow(k,3))){
09
10                         System.out.println(i+"^3"+"j+"^3"+"k+"^3"+" = "+x);
11                     }
12                 }
13             }
14         }
15     public static void main(String[] args) {
16         work();
17     }
18 }

```

Данная программа находит такие цифры А, В и С, сумма кубов которых равна числу, составленному из этих цифр.

В программе имеются три цикла, вложенные друг в друга, каждый цикл выполняется десять раз, таким образом тело цикла 07-09 выполняется одну тысячу раз (10x10x10). Переменная x , составленная из кубов счетчиков циклов, где первый счетчик играет роль сотен числа, второй десятков, третий единиц. Если полученное x равно сумме кубов счетчиков, то на консоль выводится информация о полученных цифрах и числе x . Нахождение кубов чисел осуществляется с помощью метода `pow()` класса `Math`, в качестве первого параметра выступает само число, а вторым параметром служит показатель степени `Math.pow(i,3)`. Можно использовать более знакомую и привычную запись $i*i*i$, $j*j*j$ и $k*k*k$.

Задания к работе

1. Создайте приложение, которое покажет, что для выражения $a^n + b^n = c^n$ (теорема Ферма) нет натуральных решений от 1 до 100 и $n > 2$. Убедитесь, что есть решения для $n=2$, и выведите их в консоль.

2. Вычислить выражение $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots + \frac{1}{9999} - \frac{1}{10000}$, используя оператор условия.

Лабораторная работа №5

Тема работы. Алгоритмизация и использование управляющих структур в Java.

Цель работы. Организация ветвления в Java.

Содержание работы. Рассматривается программа, в которой используется оператор множественного выбора, выполняющий задачу множественного ветвления.

Программа 1. Перевод чисел из десятичной системы счисления в двоичную и шестнадцатеричную.

```
01 public class Trans {
02     public static String fromdectobin(int x) {
03         String res="";
04         short q = 0;
05         while(x>0) {
06             q=(short) (x%2);
07             x/=2;
08             res=q+res;
09         }
10         return res;
11     }
12     public static String fromdectohex(float x) {
13         String res="";
14         short q = 0;
15         int c = 2;
16         while(c>0) {
17             q=(short) (16*(x/16-Math.floor(x/16)));
18             c=(int) Math.floor(x/16);
19             x=c;
20             if(q<10) {
21                 res=q+res;
22             }else{
23                 switch(q) {
24                     case 10:
25                         res="A"+res;
26                         break;
27                     case 11:
28                         res="B"+res;
29                         break;
30                     case 12:
31                         res="C"+res;
32                         break;
33                     case 13:
34                         res="D"+res;
35                         break;
36                     case 14:
37                         res="E"+res;
38                         break;
39                     case 15:
40                         res="F"+res;
```

```

41         break;
42     }
43 }
44 }
45     return res;
46 }
47     public static void main(String[] args) {
48         System.out.println(fromdectobin(90367));
49         System.out.println(fromdectohex(90367));
50     }
51 }

```

В данной программе приведены два метода, один переводит десятичное число в двоичную систему счисления, второй — в шестнадцатеричную.

Для перевода десятичного числа в двоичное представление используется стандартный метод — деление исходного числа на 2 с сохранением остатка, в нашем примере остаток прибавляется к строке `res` с левой стороны (08). В данном методе используется цикл с предусловием `while()`, который каждый раз получает остаток от деления на 2, затем делит исходное число на 2 и заносит полученный в первом действии остаток слева в строку. В конце, после цикла, метод возвращает полученную строку.

Способ перевода из десятичного в шестнадцатеричное представление использует аналогичный прием, производя последовательно деление на 16, за одним исключением — остаток в нашей программе вычисляется с помощью функции `floor()`, которая возвращает целую часть дробного числа. В результате получим целочисленный остаток от деления. Процесс записи остатков в строку более сложный, чем в случае двоичной системы счисления, так как в шестнадцатеричной для представления числа недостаточно базового набора цифр от 0 до 9 и необходимо использовать буквы A-F. Для того чтобы в строку результата добавлять буквы, в программе используется оператор множественного выбора `switch()`, в котором производится проверка остатка. В зависимости от величины остатка, в строку результата добавляется необходимая буква (строки 23-42).

Методы `fromdectobin`, `fromdectohex` возвращают значения типа `String`, которые можно непосредственно использовать в качестве аргумента метода `print()` для вывода на консоль (строки 48, 49). В качестве аргументов этих методов берутся числа, которые требуется перевести в заданную систему счисления.

Задания к работе

1. Для произвольной цифры от 0 до 9 вывести на консоль ее значение прописью. Например, для цифры 9 на консоли должна быть напечатана строка «Девять».

Лабораторная работа №6

Тема работы. Массивы и строки.

Цель работы. Научиться использовать массивы и строки в Java.

Содержание работы. Работа с массивами: поиск минимального элемента и сортировка массива, а также простейшие приемы работы со строками.

Программа 1. Работа с массивами

```
01 public class Arrays {
02     public static short minX() {
03         short x=0;
04         short[] array = new short[10];
05         System.out.print("Исходный массив: ");
06         for (int i=0;i<10;i++){
07             array[i]=(short)Math.round(50*Math.random());
08             System.out.print(array[i]+", ");
09         }
10         x=array[0];
11         for(int i=1;i<10;i++){
12             if(x>array[i]){
13                 x=array[i];
14             }
15         }
16         return x;
17     }
18     public static void sort(){
19         short temp;
20         short[] array = new short[10];
21         System.out.print("\nИсходный массив: ");
22         for (int i=0;i<10;i++){
23             array[i]=(short)Math.round(50*Math.random());
24             System.out.print(array[i]+", ");
25         }
26         System.out.print("\nМассив отсортированный: ");
27         for(int i=0;i<9;i++){
28             for(int j=9;j>i;j--){
29                 if(array[j-1]>array[j]){
30                     temp=array[j];
31                     array[j]=array[j-1];
32                     array[j-1]=temp;
33                 }
34             }
35             System.out.print(array[i]+", ");
36         }
37     }
38     public static void main(String[] args) {
39         System.out.print("\nМинимальный элемент: "+minX());
40         sort();
41     }
42 }
```

В данной программе используются два метода — `minX()` и `sort()`. В каждом из представленных методов описывается по одному массиву, выделяя память на десять элементов для каждого массива (04, 20). Затем происходит заполнение массивов случайными числами с помощью метода `Math.random()` (06-09, 22-25). Явным преобразованием (`short`) приводим полученные значения типа `double` к значениям типа `short` (7,23).

После того, как массивы готовы, можно производить над ними различные действия. Метод `minX()` ищет минимальный элемент в полученном массиве (11-15).

Второй метод производит сортировку полученного массива (27-34) методом пузырька. Затем выводит значения его элементов (35).

В методе `main()` вызываем `minX()` и `sort()`. Поскольку метод `minX()` возвращает значение строкового типа, его можно использовать в качестве аргумента конструкции `System.out.print()`.

Программа 2. Работа со строками

```
01 public class Strings {
02     public static String compare(String s1, String s2){
03         String s3="";
04         if (s1.equals(s2)){
05             s3="Строки \""+s1+"\" и \""+s2+"\" равны";
06         } else {
07             s3="Строки \""+s1+"\" и \""+s2+"\" не равны";
08         }
09         return s3;
10     }
11     public static String add(String s1, String s2){
12         System.out.print("\nРезультат сложения строк \""+s1+"\" и
13         "+"\""+s2+"\": ");
14         s1+=" "+s2;
15         return s1;
16     }
17     public static void main(String[] args) {
18         System.out.println(compare("АВВГ", "АВВ"));
19         System.out.print(compare("АВВ", "АВВ"));
20         System.out.print(add("Hello", "World"));
21     }
```

В приложении имеется два метода, которые сравнивают и склеивают две строки. Первый метод `compare()` производит сравнение двух строк и выводит результат на консоль. Сравнение производится с помощью функции `equals()` в строке (04). Если строки совпадают, данная функция возвращает значение `true`. Если строки равны (имеют одинаковый набор символов), выводится соответствующее сообщение (05), иначе сообщение (07). В конструкции присвоения строке `s3` (05, 07) были использованы символы Escape-последовательности, приведенные в лабораторной работе №2.

Во втором методе происходит простое склеивание строк с помощью операции «+»(13).

Задания к работе

1. Дан массив из целых чисел $A(n)$, где $n=1, 25$. Необходимо поменять местами его максимальный и минимальный элемент.
2. Дан массив из целых чисел $B(n)$, где $n=1, 25$. Необходимо упорядочить массив по возрастанию.
3. Дан массив из целых чисел $C(n)$, где $n=1, 20$. Необходимо найти среднее значение и вывести его на консоль.
4. Дан массив из целых чисел $D(n)$, где $n=1, 30$. Посчитайте сумму четных и нечетных элементов массива.
5. Напишите программу, выводящую на консоль таблицу 3×5 случайных элементов ($a(i, j) < 10$).
6. Измените программу 1 так, чтобы она выводила на консоль еще и максимальный элемент (с помощью описания нового метода, например `maxX()`).
7. Даны 5 строк $s1, s2, s3, s4$ и $s5$, на основе условия: если строка $s4$ равна строке $s5$, нужно сложить строки $s1$ и $s2$, иначе нужно сложить строки $s1$ и $s3$.

Лабораторная работа №7

Тема работы. Введение в объектно-ориентированное программирование.

Цель работы. Изучение основ объектно-ориентированного программирования на языке Java.

Содержание работы. В работе рассматриваются приложения, демонстрирующие основные принципы объектно-ориентированного программирования.

Программа 1. Инкапсуляция полей и наследование.

```
01 public class Encapsulate {
02     public int field1 = 100;
03     protected int field2 = 150;
04     int field3 = 200;
05     private int field4 = 250;
06     public int getField4() {
07         return this.field4;
08     }
09     public static void main(String[] args) {
10         new Class2().method1();
11     }
12 }
13 class Class2 extends Encapsulate {
14     public void method1() {
15         System.out.println("Значение открытого public поля
- field1: "+this.field1);
16         System.out.println("Значение защищенного protected
поля - field2: "+this.field2);
17         System.out.println("Значение поля без модификатора
- field3: "+this.field3);
18         System.out.println("Значение инкапсулированного закрытого
private поля - field4: "+getField4());
19     }
20 }
```

В приведенном примере демонстрируются два основных принципа объектно-ориентированного программирования: инкапсуляция и наследование. Создается класс Encapsulate (01-12), в котором объявляются и инициализируются поля field1, field2, field3 и field4. Каждое поле имеет различный уровень доступа, поскольку при описании использованы модификаторы доступа к элементам класса: public, protected, private. Затем описывается новый класс Class2, который наследуется от класса Encapsulate с помощью зарезервированного слова extends. В данном классе описывается новый метод, расширяющий суперкласс, в котором выводятся значения полей, описанных в суперклассе. Вы видите, что в классе Class2 нет описания этих полей, они наследуются из суперкласса Encapsulate, так же как и метод getField4(). Поскольку метод main() находится в классе Encapsulate, то для проверки работы метода method1() нужно создать экземпляр класса Class2 и вызвать этот метод, что и происходит в строке 10.

Интересно отметить особенность инкапсуляции, когда поле `filed4`, объявленное с ключевым словом `private`, не доступно в классах потомках, хотя и наследуется. Для проверки этого факта измените строку 18, заменив вызов `getField4()` на прямой запрос значения поля `this.filed4`. В данном случае Eclipse сообщит об ошибке, а именно, что поле `Encapsulate.filed4` не доступно. Принцип инкапсуляции применяется для сокрытия переменных или методов класса от внешних клиентов (классов, методов).

Программа 2. Полиморфизм.

```
01 public class Polymorph {
02     void method1(){
03         System.out.println("Это был вызван метод суперкласса
Polymorph");
04     }
05     public static void main(String[] args) {
06         new NewClass1().method1();
07         new NewClass2().method1();
08     }
09 }
10 class NewClass1 extends Polymorph {
11     void method1(){
12         System.out.println("Это был вызван метод класса потомка
NewClass1");
13     }
14 }
15 }
16 class NewClass2 extends Polymorph {
17     void method1(){
18         System.out.println("Это был вызван метод класса потомка
NewClass2");
19     }
20 }
```

В приложении демонстрируется простейший пример полиморфизма. Имеется три класса `Polymorph`, `NewClass1`, `NewClass2`, причем `Polymorph` является суперклассом по отношению к `NewClass1`, `NewClass2`. В классе `Polymorph` описан метод `method1()` (02), который выводит сообщение в консоль. Классы `NewClass1`, `NewClass2` наследуют этот метод и переопределяют его (`override`), осуществляя тем самым перегрузку метода. Далее в методе `main()` суперкласса создается два объекта экземпляра классов.

В результате оба класса наследуют один и тот же метод, но выполняется он у каждого из них по-разному или в разной форме. Таким образом, вызывая один и тот же метод, с помощью разных объектов можно получить разные по форме результаты, тогда более понятным становится смысл термина «полиморфизм».

Задания к работе

1. Создайте приложение, в котором имеются три класса: `fclass1`, `fclass2`, `fclass3`. В классе `fclass1` содержится метод `main()`. Класс `fclass3` наследуется от `fclass1`, а `fclass2` — от `fclass3`.

2. Создайте приложение, в котором имеются два класса: `fclass1`, `fclass2`. В классе `fclass1` содержится метод `main()`. Создайте методы в `fclass2` для доступа к закрытым переменным класса `fclass1`.

Лабораторная работа №8

Тема работы. Работа с графическим интерфейсом пользователя. Обработка событий.

Цель работы. Знакомство с элементами разработки графического интерфейса пользователя. Создание приложения из нескольких файлов.

Содержание работы. В данной работе рассматриваются программы, демонстрирующие основы работы с графическим интерфейсом пользователя и обработкой событий компонентов ГИП.

В данной работе для создания программы графопостроителя нам понадобится четыре класса, которые будут находиться в разных файлах. Для того, чтобы в дальнейшем избежать путаницы, необходимо данные файлы поместить в отдельную папку (или, в интерпретации платформы Eclipse, в отдельный подпакет). Создайте новую папку, щелкнув правой кнопкой мыши на имени проекта в представлении Package explorer, и в раскрывшемся меню (рис. 15), выбрав пункт «Folder». Назовите ее «MyGraph». Затем в данной папке аналогично создайте четыре новых класса:

- «Graphic» — главный класс, в котором будет метод `main()`.
- «Sinus» — класс графика синуса.
- «Cosinus» — класс графика косинуса.
- «X2» — класс графика параболы.

Помните, если классы описаны с любым из модификаторов видимости `public`, `protected` или `private`, то имена классов должны совпадать с именами файлов, в которых хранятся данные классы.

После того, как классы будут созданы, введите предложенный ниже код программы.

Программа 1. Построение графиков элементарных функций.

Класс Graphic

```
01 package MyGraph;
02 import java.awt.*;
03 import java.awt.event.*;
04 import javax.swing.*;
05 public class Graphic extends JFrame{
06     Graphic(String s){
07         super(s);
08         setLayout(null);
09         setSize(120,200);
10         setVisible(true);
11         this.setDefaultCloseOperation(EXIT_ON_CLOSE);
12         this.setResizable(false);
13         Button sin = new Button("Sin");
14         sin.setBounds(5, 20, 100, 25);
15         add(sin);
16         Button cos = new Button("Cos");
17         cos.setBounds(5, 70, 100, 25);
18         add(cos);
19         Button x2 = new Button("Парабола");
20         x2.setBounds(5, 120, 100, 25);
```

```

21     add(x2);
22     sin.addActionListener(new ActionListener() {
23         public void actionPerformed(ActionEvent event) {
24             new Sinus("Синус");
25         }
26     });
27     cos.addActionListener(new ActionListener() {
28         public void actionPerformed(ActionEvent event) {
29             new Cosinus("Косинус");
30         }
31     });
32     x2.addActionListener(new ActionListener() {
33         public void actionPerformed(ActionEvent event) {
34             new X2("Парабола");
35         }
36     });
37 }
38 public static void main(String[] args) {
39     new Graphic("Графопостроитель");
40 }
41 }

```

Класс Sinus

```

01 package MyGraph;
02 import java.awt.Color;
03 import java.awt.Graphics;
04 import javax.swing.JFrame;
05 public class Sinus extends JFrame {
06     Sinus(String s) {
07         super(s);
08         setLayout(null);
09         setSize(600, 300);
10         setVisible(true);
11         this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
12         this.setResizable(false);
13         this.setLocation(100, 100);
14     }
15     public void paint(Graphics gr) {
16         int y; int j=0; int k=0;
17         gr.setColor(Color.WHITE);
18         gr.fillRect(0, 0, 600, 300);
19         gr.setColor(Color.lightGray);
20         while(j<600) {
21             gr.drawLine(j, 0, j, 300);
22             j+=30;
23         }
24         while(k<300) {
25             gr.drawLine(0, k, 600, k);
26             k+=30;
27         }

```

```

28     gr.setColor(Color.BLACK);
29     gr.drawLine(300, 0, 300, 300);
30     gr.drawLine(0, 150, 600, 150);
31     gr.drawLine(120, 140, 120, 160);
32     gr.drawLine(480, 140, 480, 160);
33     gr.drawString("0", 305, 165);
34     gr.drawString("-"+"\u03c0", 125, 140);
35     gr.drawString("\u03c0", 485, 140);
36     gr.setColor(Color.RED);
37     for(double i=0;i<1000;i++){
38         y=(int)(80*Math.sin(Math.PI*i/180));
39         gr.drawLine((int)i-240, y+150, (int)i-240, y+150);
40     }
41     gr.dispose();
42 }
43 }

```

Класс Cosinus

```

01 package MyGraph;
02 import java.awt.Color;
03 import java.awt.Graphics;
04 import javax.swing.JFrame;
05 public class Cosinus extends JFrame{
06     Cosinus(String s){
07         super(s);
08         setLayout(null);
09         setSize(600,300);
10         setVisible(true);
11         this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
12         this.setResizable(false);
13         this.setLocation(200, 200);
14     }
15     public void paint(Graphics gr){
16         int y; int j=0; int k=0;
17         gr.setColor(Color.WHITE);
18         gr.fillRect(0, 0, 600, 300);
19         gr.setColor(Color.lightGray);
20         while(j<600){
21             gr.drawLine(j, 0, j, 300);
22             j+=30;
23         }
24         while(k<300){
25             gr.drawLine(0, k, 600, k);
26             k+=30;
27         }
28         gr.setColor(Color.BLACK);
29         gr.drawLine(300, 0, 300, 300);
30         gr.drawLine(0, 150, 600, 150);
31         gr.drawLine(120, 140, 120, 160);
32         gr.drawLine(480, 140, 480, 160);

```

```

33     gr.drawString("0", 305, 165);
34     gr.drawString("-"+"\u03c0", 125, 140);
35     gr.drawString("\u03c0", 485, 140);
36     gr.setColor(Color.RED);
37     for(double i=0;i<1000;i++){
38         y=(int) (80*Math.cos(Math.PI*i/180));
39         gr.drawLine((int)i-240, y+150, (int)i-240, y+150);
40     }
41     gr.dispose();
42 }
43 }

```

Класс X2

```

01 package MyGraph;
02 import java.awt.Color;
03 import java.awt.Graphics;
04 import javax.swing.JFrame;
05 public class X2 extends JFrame{
06     X2(String s){
07         super(s);
08         setLayout(null);
09         setSize(600,300);
10         setVisible(true);
11         this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
12         this.setResizable(false);
13         this.setLocation(300, 300);
14     }
15     public void paint(Graphics gr){
16         int y; int j=0; int k=0;
17         gr.setColor(Color.WHITE);
18         gr.fillRect(0, 0, 600, 300);
19         gr.setColor(Color.lightGray);
20         while(j<600){
21             gr.drawLine(j, 0, j, 300);
22             j+=50;
23         }
24         while(k<300){
25             gr.drawLine(0, k, 600, k);
26             k+=50;
27         }
28         gr.setColor(Color.BLACK);
29         gr.drawLine(300, 0, 300, 300);
30         gr.drawLine(0, 150, 600, 150);
31         gr.drawString("0", 305, 165);
32         gr.setColor(Color.RED);
33         for(double i=0;i<1000;i++){
34             y=- (int) (i*i/300)+150;
35             gr.drawLine((int)i+300, y, (int)i+300, y);
36             gr.drawLine(-(int)i+300, y, -(int)i+300, y);
37         }

```

```

38 |     gr.dispose();
39 | }
40 | }

```

В классе Graphics описано окно размером 120x200, в котором расположены три кнопки. Нажатие кнопки обслуживается (обрабатывается) специальной процедурой, которая называется процедурой обработки события ActionEvent. При нажатии кнопки происходит создание новой формы, в которой будет построен соответствующий график элементарной функции (22-26, 27-31, 32-36).

Классы обработчиков событий описаны внутри параметра метода добавления слушателя addActionListener() (22, 27, 32). Интересно, что параметром этого метода является новый объект интерфейса ActionListener с одним методом actionPerformed(), который и отвечает за поведение программы, в случае если будет вызвано данное событие (нажатие на кнопки).

В классах, которые производят построение графиков, в строках 20-27 с использованием двух циклов происходит построение координатной сетки светло-серым цветом, затем черным цветом строятся оси абсцисс и ординат.

Наибольший интерес представляют строки 37-40 (для синуса и косинуса) и 33-37 (для параболы) в которых происходит рисование графиков. Синус и косинус рисуются по аналогии: вначале вычисляется значение y и методом drawString() рисуется «линия» длиной в один пиксел. Координата y вычисляется путем явного преобразования типа double в тип int, полученного в результате выполнения выражения $80 * \text{Math.cos}(\text{Math.PI} * i / 180)$.

Парабола строится аналогичным способом, в два этапа. На первом этапе строится положительная часть параболы и далее отрицательная.

Программа 2. Надписи на кнопках.

```

01 | import java.awt.*;
02 | import java.awt.event.*;
03 | import javax.swing.*;
04 | public class ExtEventHandler extends JFrame {
05 |     ExtEventHandler(String s){
06 |         super(s);
07 |         setLayout(null);
08 |         setSize(100,200);
09 |         setVisible(true);
10 |         setResizable(false);
11 |         setDefaultCloseOperation(EXIT_ON_CLOSE);
12 |         Button b1 = new Button("Первая кнопка");
13 |         b1.setBounds(2, 5, 96, 22);
14 |         add(b1);
15 |         Button b2 = new Button("Вторая кнопка");
16 |         b2.setBounds(2, 100, 96, 22);
17 |         add(b2);
18 |         b1.addActionListener(new Handler(b1, b2));
19 |         b2.addActionListener(new Handler(b1, b2));

```

```

20     }
21     public static void main(String[] args) {
22         new ExtEventHandler("");
23     }
24 }
25 class Handler implements ActionListener{
26     private Button ba;
27     private Button bb;
28     String temp;
29     Handler(Button b1, Button b2){
30         this.ba=b1;
31         this.bb=b2;
32     }
33     public void actionPerformed(ActionEvent e) {
34         temp = ba.getLabel();
35         ba.setLabel(bb.getLabel());
36         bb.setLabel(temp);
37     }
38 }

```

В классе ExtEventHandler (04-24) приводится основное описание интерфейса пользователя: форма размером 100x200 и две кнопки с надписями: первая кнопка (12-14) и вторая кнопка (15-17).

В приложении для обработки событий, которые возникают при нажатии на кнопки, описан отдельный класс Handler (25-38), который использует интерфейс ActionListener.

Поскольку в описании ActionListener нельзя напрямую использовать компоненты, описанные в конструкторе класса ExtEventHandler, то нужно создать ссылки на объекты, которые будут использованы. Создание ссылочных объектов производится в строках 26, 27, а сами ссылки создаются в конструкторе (30, 31). Когда ссылки готовы, в процессе добавления слушателей (18, 19), в параметрах указываем имена реальных компонентов.

Таким образом, при нажатии на любую из двух кнопок меняются местами их надписи (34-36).

Задания к работе

1. Создайте форму размером 500x500 и нарисуйте на ней домик.
2. Сумматор. Создайте приложение, которое представляет собой форму 200x150. На форме разместите три текстовых поля и одну кнопку с надписью «Расчет». При нажатии на кнопку значения, введенные в первые два текстовых поля, складываются и результат записывается в третье поле.

Лабораторная работа №9

Тема работы. Апплеты.

Цель работы. Знакомство с элементами разработки Java-апплетов.

Содержание работы. Разработка апплетов.

Программа 1. Простейший апплет.

```
01 import java.awt.*;
02 import java.applet.*;
03 public class My_Applet extends Applet{
04     public void paint(Graphics g){
05         g.drawString("Hello, World!", 5, 30);
06     }
07 }
```

Приложение представляет собой простейший Java-апплет, в котором выводится строка с использованием ранее рассмотренного метода (05) `drawString()`.

Для запуска и просмотра апплетов в Eclipse используется специальное программное средство `AppletViewer`.

Программа 2. Использование элементов графического интерфейса пользователя в апплетах.

```
01 import java.awt.*;
02 import java.awt.event.ActionEvent;
03 import java.awt.event.ActionListener;
04 import java.applet.*;
05 public class My_Applet extends Applet{
06     public void init(){
07         setLayout(null);
08         final TextField tf1 = new TextField(15);
09         tf1.setBounds(1, 1, 100, 20);
10         add(tf1);
11         final TextField tf2 = new TextField(15);
12         tf2.setBounds(1, 25, 100, 20);
13         add(tf2);
14         final TextField tf3 = new TextField(15);
15         tf3.setBounds(1, 50, 100, 20);
16         add(tf3);
17         Button b1 = new Button("Кнопка 1");
18         b1.setBounds(1, 100, 100, 20);
19         add(b1);
20         b1.addActionListener(new ActionListener(){
21             public void actionPerformed(ActionEvent event){
22                 try{
23                     tf3.setText(String.valueOf((Long.valueOf(tf1.getText())+Long.v
24                         alueOf(tf2.getText()))));
25                     tf3.setText("Введите числа");
26                 }
27             }
28         });
```

```
29 |   }  
30 | }
```

В данной программе описан апплет, в котором расположены три текстовых поля и одна кнопка. При нажатии на кнопку введенные численные значения первых двух полей складываются и результат записывается в третье поле (вторая задача из лабораторной работы №8).

В отличие от обычных приложений Java, в апплетах инициализация и добавление компонентов графического интерфейса пользователя осуществляется не в конструкторе, а в методе `init()`.

Задания к работе

1. Выполните задание 1 из лабораторной работы №8 в форме апплета.
2. Создайте апплет, в котором при нажатии на кнопку выводится матрица 3x3. Используйте метод `drawString()`.

Лабораторная работа №10

Тема работы. Приложение «Калькулятор».

Цель работы. Знакомство со сложным приложением Java.

Содержание работы. Анализ и подробный разбор кода. Запуск и тестирование приложения.

Программа 1. Калькулятор.

```
01 import java.awt.*; // Импорт пакета awt
02 import java.awt.event.*; // Импорт пакета обработки событий
03 import javax.swing.*; // Импорт пакета swing
04 public class Calc extends JFrame{ // Начало класса Calc, наслед.
    от класса JFrame
05 double temp = 0; // Объявление переменной temp типа double
06 Char op = ''; // Объявление переменной op типа char
07 Calc(String s){ // Начало определения конструктора Calc
08     super(s); // Вызов конструктора суперкласса (JFrame)
09     setLayout(null); // Отказ от менеджера размещения
10     setSize(250,250); // Установка размеров окна
11     setVisible(true); // Установка видимости на экране
12     this.setDefaultCloseOperation(EXIT_ON_CLOSE); //
    Установка по умолчанию для закрытия окна
13     final TextField display = new TextField(""); // Создание
    объекта типа TextField с именем display (Поле ввода)
14     display.setEditable(false); // Запрещение редактирования
15     display.setBounds(2, 2, 238, 22); // Установка
    расположения и размеров поля ввода
16     add(display); // Добавление поля ввода в контейнер окна
17     Button b1 = new Button("1"); // Строки 17-61 описывают
18     b1.setBounds(2, 30, 40, 40); // создание 15 кнопок:
19     add(b1); // 10 кнопок для ввода цифр,
20     Button b2 = new Button("2"); // 4 кнопки для основных
21     b2.setBounds(52, 30, 40, 40); // арифметических операций и
22     add(b2); // одна кнопка для расчета
23     Button b3 = new Button("3"); // результата. После
24     b3.setBounds(102, 30, 40, 40); // создания каждой из кнопок
25     add(b3); // устанавливаются их
26     Button b4 = new Button("4"); // местоположения и размеры.
27     b4.setBounds(2, 80, 40, 40); // Затем кнопки добавляются
28     add(b4); // в контейнер окна.
29     Button b5 = new Button("5");
30     b5.setBounds(52, 80, 40, 40);
31     add(b5);
32     Button b6 = new Button("6");
33     b6.setBounds(102, 80, 40, 40);
34     add(b6);
35     Button b7 = new Button("7");
36     b7.setBounds(2, 130, 40, 40);
37     add(b7);
38     Button b8 = new Button("8");
```

```

39         b8.setBounds(52,130,40,40);
40         add(b8);
41         Button b9 = new Button("9");
42         b9.setBounds(102,130,40,40);
43         add(b9);
44         Button b0 = new Button("0");
45         b0.setBounds(2,180,40,40);
46         add(b0);
47         Button beq = new Button("=");
48         beq.setBounds(52,180,90,40);
49         add(beq);
50         Button bplus = new
Button("+");
51         bplus.setBounds(152,30,80,40);
52         add(bplus);
53         Button bminus = new Button("-
");
54         bminus.setBounds(152,80,80,40);
55         add(bminus);
56         Button bmul = new Button("*");
57         bmul.setBounds(152,130,80,40);
58         add(bmul);
59         Button bdiv = new Button("/");
60         bdiv.setBounds(152,180,80,40);
61         add(bdiv);
62         b1.addActionListener(new ActionListener()
{
63         public void actionPerformed(ActionEvent
event) {
64         display.setText(display.getText()+"1");
65         }
66         });
67         b2.addActionListener(new ActionListener()
{
68         public void actionPerformed(ActionEvent
event) {
69         display.setText(display.getText()+"2");
70         }
71         });
72         b3.addActionListener(new ActionListener()
{
73         public void actionPerformed(ActionEvent
event) {
74         display.setText(display.getText()+"3");
75         }
76         });
77         b4.addActionListener(new ActionListener()
{
78         public void actionPerformed(ActionEvent
event) {

```

В строках 62-140 для каждой из кнопок производится обработка событий на нажатие. Для цифровых кнопок – добавление к тексту поля ввода соответствующей цифры, для арифметических операций – установка «флага» ор соответствующей операции, сохранение в переменной temp введенного в поле display числа, путем преобразования строки в число и очистка дисплея.

```

79     display.setText(display.getText()+"4");
80     }
81     });
82     b5.addActionListener(new ActionListener()
83     {
84     public void actionPerformed(ActionEvent
85     event) {
86     display.setText(display.getText()+"5");
87     }
88     });
89     b6.addActionListener(new ActionListener()
90     {
91     public void actionPerformed(ActionEvent
92     event) {
93     display.setText(display.getText()+"6");
94     }
95     });
96     b7.addActionListener(new ActionListener()
97     {
98     public void actionPerformed(ActionEvent
99     event) {
100    display.setText(display.getText()+"7");
101    }
102    });
103    b8.addActionListener(new ActionListener()
104    {
105    public void actionPerformed(ActionEvent
106    event) {
107    display.setText(display.getText()+"8");
108    }
109    });
110    b9.addActionListener(new ActionListener()
111    {
112    public void actionPerformed(ActionEvent
113    event) {
114    display.setText(display.getText()+"9");
115    }
116    });
117    b0.addActionListener(new ActionListener()
118    {
119    public void actionPerformed(ActionEvent
120    event) {
121    display.setText(display.getText()+"0");
122    }
123    });
124    bplus.addActionListener(new
125    ActionListener() {
126    public void actionPerformed(ActionEvent
127    event) {
128    op = '+';
129    temp =

```

```

117     Double.valueOf(display.getText());
118     display.setText("");
119     });
120     bminus.addActionListener(new
ActionListener() {
121     public void actionPerformed(ActionEvent
event) {
122         op = '-';
123         temp =
Double.valueOf(display.getText());
124         display.setText("");
125     }
126     });
127     bdiv.addActionListener(new
ActionListener() {
128     public void actionPerformed(ActionEvent
event) {
129         op = '/';
130         temp =
Double.valueOf(display.getText());
131         display.setText("");
132     }
133     });
134     bmul.addActionListener(new
ActionListener() {
135     public void actionPerformed(ActionEvent
event) {
136         op = '*';
137         temp =
Double.valueOf(display.getText());
138         display.setText("");
139     }
140     });
141     beq.addActionListener(new ActionListener()
{
142     public void actionPerformed(ActionEvent
event) {
143         switch(op) {
144         case '+':
display.setText(String.valueOf(temp+Double.v
alueOf(display.getText()))); break;
145         case '-':
display.setText(String.valueOf(temp-
Double.valueOf(display.getText()))); break;
146         case '*':
display.setText(String.valueOf(temp*Double.v
alueOf(display.getText()))); break;
147         case '/':
display.setText(String.valueOf(temp/Double.v
alueOf(display.getText()))); break;

```

В строках 141-151 происходит обработка события нажатия на кнопку «=». Путем перебора значений флага op, определяется, какая операция была выбрана и происходит сложение значения переменной temp со значением, введенным в поле после нажатия одной из кнопок

```

148     }
149     }
150     });
151
152     } // Завершение конструктора Calc
153     public static void main(String[] args) { // Начало метода main()
154     new Calc("Calculator"); // Создание объекта Calc
155     } // Завершение метода main()
156 } // Завершение класса Calc

```

арифметических
операций.

Задания к работе

1. Измените данную программу так, чтобы числа можно было бы вводить в поле с клавиатуры, при этом перехватывая возможные исключения типа `NumberFormatException`.
2. Добавьте к данному калькулятору две кнопки — «Память» и «Вызов из памяти», с помощью которых можно будет сохранить в память числовое значение, находящееся в поле, и вызвать из памяти ранее сохраненное значение.

Глоссарий

А

AWT — **Abstract Windowing Toolkit** — пакет (библиотека) базовых компонентов графического интерфейса пользователя.

Абстракция — определение, характеризующее уровень обобщенности описания того или иного объекта или процесса.

Абстрактный класс (abstract class) — класс, который содержит один или более абстрактных методов, вследствие чего нельзя создавать экземпляры данного класса. Абстрактные классы определены таким образом, чтобы другие классы могли расширять и конкретизировать их, реализуя абстрактные методы.

Абстрактный метод (abstract method) — метод, не имеющий реализации.

Аргумент — элемент некоторых данных, указываемый при вызове метода.

Array (массив) — см. массив.

Б

Байт-код (bytecode) — машинно-независимый код, генерируемый Java-компилятором и исполняемый Java-интерпретатором.

Байт (byte) — последовательность из восьми битов. В языке программирования Java определен соответствующий тип byte.

В

Виртуальная машина JAVA — программный "механизм выполнения", который безопасно выполняет байт-коды файлов классов Java на микропроцессоре (компьютера или другого электронного устройства).

Возврат (return) — оператор возвращения одного из типов значений методом класса.

Выражение — конструкция, состоящая из данных и операторов, возвращающее некоторое значение в качестве результата.

Г

GUI — см. ГИП.

ГИП — графический интерфейс пользователя — средство организации программного графического интерфейса средствами библиотек AWT и SWING.

Д

Динамическая переменная — переменная, наследуемая в потомках, становящаяся отдельным членом каждого экземпляра, не зависящая от тех же переменных других экземпляров.

Декларация — процесс объявления новых членов класса.

Декремент — оператор, увеличивающий значение операнда на единицу.

E

Event — см. событие.

I

Инкапсуляция — локализация части данных в пределах класса. Поскольку объекты инкапсулируют данные и реализацию, пользователь может рассматривать объект как черный ящик, предоставляющий услуги. Переменные и методы экземпляров класса могут добавляться, удаляться или изменяться, но до тех пор, пока услуги, предоставляемые объектом, не изменяются, нет необходимости переписывать код, использующий данный объект. См. также переменная экземпляра класса и метод экземпляра класса.

Инкремент — оператор, уменьшающий значение операнда на единицу.

Inheritable — см. наследование.

Иерархия — классификация соотношений, в которой каждый элемент, кроме верхнего (называемого корнем), является специализированным видом элемента, расположенного над ним. Каждый элемент может иметь один или несколько элементов, находящихся ниже него в иерархии. В иерархии классов Java, образуемой при наследовании, корнем является класс Object.

Имя — символьная или символьно-цифровая идентификация какого-либо члена класса или самого класса, используемая в качестве вызова.

Идентификатор — имя объекта для компилятора, используемое для связывания в процессе компиляции программы.

Интерфейс — понятие, используемое для определения набора методов и постоянных значений (класса специального вида). Интерфейс в дальнейшем может реализовываться классами, которые определяют этот интерфейс с ключевым словом `implements`.

ИЛИ — логический оператор, возвращающий значение «истина» при условии наличия хотя бы одного из операндов выражения, имеющего значение «истина».

Исключающее ИЛИ — логический оператор, возвращающий значение «истина» при условии различных логических значений двух операндов выражения.

Исключительная ситуация — ситуация, возникающая во время работы программы вследствие неудачного, или в случае невозможности выполнения операции, или запрограммированная с использованием оператора `throw`. Если в программе не предусмотрена реакция на возникшую ситуацию, дальнейшее исполнение программы невозможно.

Источник события — объект, чаще компонент ГИП, являющийся возбудителем некоторого события.

K

Класс — тип в языке программирования Java, определяющий реализацию особого вида объекта. Описание класса определяет экземпляр класса, его переменные и методы.

Компонент — элемент библиотеки графического интерфейса пользователя AWT или SWING.

Код — исходный текст программы или класса.

Комментарий — часть кода, предваряемая специальным символом (в JAVA это символ //), которую компилятор при интерпретации игнорирует.

Константа — член класса, который на протяжении всей программы не может быть изменен, в том числе и в потомках и экземплярах класса. Задается модификатором `final`.

Конструктор — особый метод, имеющийся в каждом классе, осуществляющий процесс создания экземпляров данного класса. В случае отсутствия явного описания, используется конструктор суперкласса.

Контейнер — сущность, обеспечивающая управление, безопасность, разработку и сервисы выполнения компонент.

Л

Логический оператор — оператор, выполняющий действие с операндами логического типа.

Логическая переменная/константа — член класса, имеющий логический тип.

Логическое выражение — совокупность операторов и операндов, возвращающая в результате логическое значение.

Логическая ошибка — ошибка программиста, характеризующаяся неверной постановкой вычислительной задачи в программе.

Локальная переменная — переменная класса, которая не участвует в наследовании и конструкции экземпляров. Задается модификатором `static`.

М

Массив (array) — совокупность элементов данных одного типа, в которой позиция каждого элемента однозначно определена целым числом (индексом массива).

Модульность — один из принципов объектно-ориентированного программирования, предполагающий, что каждый класс должен составлять отдельный модуль, а члены класса, обращение к которым не планируется в дальнейшем, должны быть *инкапсулированы*.

Метод — функция, определенная в классе. Пока не оговорено обратное, метод не является статическим. Различают методы экземпляров и методы классов. Методы классов — локальные методы, как и переменные, задаются модификатором `static`.

Модификатор — элемент языка объектно-ориентированного программирования, задающий параметры декларируемых членов.

Н

Наследование — концепция классов, автоматически включающих все переменные и методы, определенные в супертипе.

Name — см. имя.

О

Объект — основной компоновочный блок объектно-ориентированных программ. Каждый объект программного модуля состоит из данных (переменные экземпляра) и функциональных возможностей (методы экземпляра).

Объектно-ориентированное программирование — метод проектирования программного обеспечения, позволяющий моделировать абстрактные или реальные объекты при помощи классов и объектов.

Окно — элемент графического интерфейса пользователя, являющийся контейнером, имеющий возможность помещать в себя другие компоненты.

Отрицание — логический процесс изменения состояния логического выражения, переменной или константы на обратное.

Оператор — элемент, использующийся в процессе вычисления выражений.

Объявление — процесс присваивания начального значения переменной или константе.

Обработка события — некоторый набор операторов, процедур и функций, выполняющийся при наступлении события.

П

Переменная — элемент данных, имеющий идентифицирующее его имя.

Подкласс — класс, наследованный от некоторого суперкласса.

Поле — элемент класса. Пока не определено обратное, поле не является статическим.

Полиморфизм — один из базовых принципов ООП, опирающийся на наследование и предполагающий наличие нескольких объектов, наследованных от одного и того же класса, но имеющих разное содержание.

Р

Рабочий стол — часть графического интерфейса пользователя оболочки программирования, отображающая совокупность элементов управления.

С

Символ — элемент алфавита, а также элемент данных текстового типа.

Событие — элемент класса Event.

Ссылка — объект, являющийся сопоставлением некоторому идентификатору некоторой области памяти, содержащей данные.

Строка — элемент данных, представляющий собой набор символов.

Суперкласс — класс, являющийся родителем одного или нескольких подклассов. Класс Object является суперклассом для любого класса.

Т

Тип — параметр данных, характеризующий их принадлежность к определенному множеству.

У

Условие — некоторый блок программы, содержащий выражение, переменную или константу, имеющий логический тип и являющийся основой выполнения какого-либо блока задач. Используется, в основном, в циклах и операторе ветвления.

Управляющие конструкции — разновидность операторов языка программирования, функции которых заключаются в управлении ходом программы.

Ф

Фокус — определение, характеризующее текущее состояние компонента. Компоненты, имеющие фокус, могут получать команды от пользователя с клавиатуры.

Ч

Член класса — метод, константа или переменная, описанная в данном классе.

Э

Экземпляр — объект, созданный на основе некоторого класса.

Список литературы

Использованная:

1. *Барнет Э.* Eclipse IDE Карманный справочник: Пер. с англ. — М.:КУДИЦ-ОБРАЗ, 2006. — 160 с.
2. *Хабибуллин И. Ш.* Самоучитель Java 2. — СПб.: БХВ-Петербург, 2007. — 720 с.
3. *Шилдт Г.* Полный справочник по Java. — М.: Вильямс, 2007. — 1040 с.
4. *Ноутон П., Шилдт Г.* Java 2. — СПб.: ВHV-Санкт-Петербург, 2008. — 1072 с.

Электронные издания

5. *Монахов В.В.* Материалы курсов, разработанных в рамках программы Sun Microsystems Teaching Grants в 2006 году — «Язык программирования Java». СПбГУ, 2006.
6. Материалы интернет-сайта <http://ru.sun.com>

Рекомендуемая:

7. *Фишер Т.* Java. Карманный справочник. — М.: Вильямс, 2008. — 224 с.
8. *Хемрадждани А.* Гибкая разработка приложений на Java с помощью Spring, Hibernate и Eclipse. — М.: Вильямс, 2008. — 352 с.
9. *Хабибуллин И. Ш.* Самоучитель Java 2. — СПб.: БХВ-Петербург, 2007. — 720 с.
10. *Шилдт Г.* Swing. Руководство для начинающих. — М.: Вильямс, 2007. — 704 с.
11. *Барнет Э.* Eclipse IDE Карманный справочник: Пер. с англ. — М.:КУДИЦ-ОБРАЗ, 2006. — 160 с.