Введение в практику работы на Java

Часть 1. Что такое Java и как она работает?

Если кратко, то Java - это один из языков программирования. Он разработан компанией Sun, и является платформо-независимым. Это означает, что программа, написанная на Java, будет одинаково выполнятся и под Windows, и под Linux, и под другими ОС. Достигается это следующим образом - текст программы переводится с помощью компилятора не в родные для процессора команды, а в коды виртуальной java-машины (JVM). Коды виртуальной машины одинаковы на любой платформе, и именно поэтому одна и та же программа и будет работать на разных платформах. Коды эти, кстати, называются байт-кодами. Сама же виртуальная машина от платформы, естественно, зависит - виртуальная машина для Windows отличается от виртуальной машины для других ОС. Мы в наших Частьах будем рассматривать создание программ на Java для Windows. Но это не означает, что они не будут работать на других платформах - как раз наоборот.

Существует два основных вида программ на Java - собственно Java-программы и апплеты. Первые выполняются как самостоятельные программы, вторые выполняются в браузере. В настоящее время почти все браузеры имеют в своем распряжении JVM. Слово почти означает, что Internet Explorer 6.0 не поддерживает JVM, но вы можете использовать продукты третьих фирм для исправления этого. У Microsoft вообще особая позиция по Java - сначала Microsoft поддерживала этот язык, но затем отказалась. Сейчас Microsoft активно продвигает свой новый язык C# и платформу .NET как альтернативу Java. В свое время Microsoft даже проиграла компании SUN судебное разбирательство по поводу нарушения лицензионного соглашения по Java - реализация Java у Microsoft была привязана к платформе Windows.

Часть 2. Что такое JDK и как его установить?

JDK расшифровывается как Java Developer Kit. Это набор программ и утилит, предназначенный для программирования на Java. В него ряд утилит. Вот некоторые из них:

- Компилятор javac. Именно он и переводит текст программы на Java в байт-коды виртуальной машины.
- Интерпретатор java. Вы с его помощью будете запускать откомпилированные в байт-коды программы. Он содержит в себе JVM (Виртуальную машину Java).
- Утилита appletviewer. С ее помощью можно запускать созданные вами апплеты. Фактически она представляет из себя браузер, который может запускать только апплеты.
- Утилита javadoc. Она предназначена для создания документации.

Есть еще и другие утилиты, но пока они нам не особеннно нужны, так что обсуждать мы их не будем.

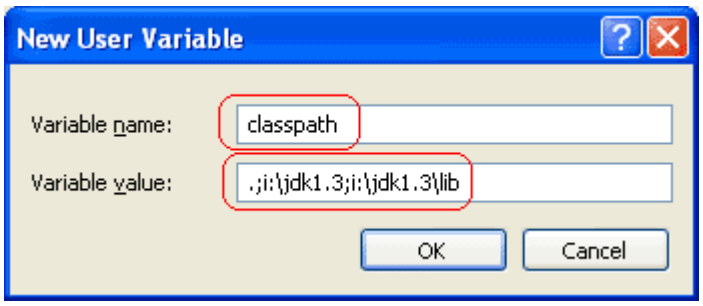
JDK - это бесплатный набор. Как следствие, вам придется работать без особого комфорта - тексты программ надо будет набирать в Блокноте или аналогичном текстовом редакторе. Скачать JDK можно с сайта компании Sun. Текущая версия - 1.3, хотя на момент написания этих строк и версия 1.4 уже не за горами. В терминах Sun нужная нам версия имеет номер 2 (т. е. с сайта Sun вы должны качать Java 2). Перед скачиванием убедитесь, что нашли JDK для нужной платформы (Windows в нашем случае).

После скачивания просто распакуйте полученный архив в папку на вашем компьютере. Лучше всего эту папку назвать jdk1.3 и расположить в корневой папке, хотя название и расположение для нее могут быть любыми. После распаковки в папке jdk1.3 появится целый ряд подпапок (bin, include, lib и другие). Теперь осталось прописать путь для нахождения соответствующих файлов. Для этого для Windows 95/98 и последующих добавьте следующие строчки в файл autoexec.bat:

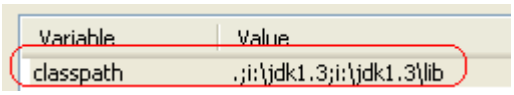
```
...
set path=c:\jdk1.3\bin
set classpath=c:\jdk1.3\lib
...
```

Разумеется, вы должны указать используемый вами путь и диск. Для вступления изменений в силу перезагрузите компьютер.

В Windows NT и ее потомках действовать надо немного по-другому. Вот так, например, вы должны действовать в Windows XP. Для установки соответствующих переменных окружения щелкните правой кнопкой мыши на рабочем столе на иконке Мой компьютер. В контекстном меню выберите Properties и перейдите на вкладку Advanced. В самом низу этой вкладки нажмите на кнопку Environment Variables. В появившемся диалоговом окне Environment Variables нажмите на кнопку New для добавления новых переменных окружения. Для добавления переменной classpath заполните поля Variable name (classpath) и Variable value (i:\jdk1.3\lib)



Нажмите на OK. Добавленная переменная появится в списке переменных:



Аналогично добавьте переменную path со значением i:\jdk1.3\bin (укажите, разумеется, значение, используемое вами).

Перезагрузите компьютер.

С установкий JDK все!

Часть 3. Первая программа на Java

Первая программа, по давно укоренившейся традиции, будет HelloWorld. Ниже приводится ее текст, который надо набрать в любом текстовом редакторе, позволяющем сохранять документ в ASCII-кодах (лучше всего для этих целей подходит Блокнот). Наберите следующий текст и сохраните его в файле HelloWorld.java:

```
class HelloWorld{
    public static void main(String [] args){
        System.out.println("Hello World!");
    }
}
```

Несколько слов по тексту программы. Во-первых, обратите внимание на слово class. Любая программа на Java использует классы (их мы начнем обсуждать через несколько Частьов), и эта - не исключение. Во-вторых, обязательно должен быть метод (функция) main. Именно с него все и начинается. В нашем случае main имеет несколько модификаторов. Модификатор public означает, что данный метод будет виден снаружи класса (подробности в последующих Частьах). Модификатор static приблизительно означает, что метод main можно вызывать не для конкретного экземпляра класса, а в самом начале программы, когда никакого экземпляра класса еще нет. И, наконец void означает, что метод main не возвращает никакого значения. В строке

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

вызывается метод println, который и выводит на экран соответствующую надпись. Этот метод берется из пространства имен System.out.

Еще одно замечание. Java, как и все C-подобные языки, различает строчные и прописные буквы. Так что, например, HelloWorld и helloworld - разные вещи.

После того, как текст набран (напомним, что мы сохраняем его в файле с именем HelloWorld.java), его надо откомпилировать. Для этого в командной строке (Для ее вызова в Windows 2000 и XP выберите Start->Run и затем наберите cmd) перейдите в папку с нашим файлом и наберите javac HelloWorld.java:

Буковка с в конце слова javac - это от английского слова компилятор. Если все было сделано правильно, то никаких сообщений выдасться не должно, а в нашей папке должен появиться еще один файл HelloWorld.class. Именно он и представляет из себя откомпилированную в байт-коды нашу программу. Для его запуска набираем в командной строке java HelloWorld:

Обратите внимание, что имя файла мы набираем без расширения.

Результатом выполнения программы будет, как и ожидалось, вывод на экран слов "Hello World!":

С первой программой все!

Часть 4. Числовые типы данных

Числовые типы данных Java перечислены в следующей таблице:

Тип	Описание	Количество байтов
int	целый	4
float	вещественный	4
char	символьный	2
short	короткое целое	2
long	длинное целое	8
double	длинное вещественное	8
byte	байт	1

Объвление переменных происходит следующим образом:

```
int a; //Переменная а целого типа
float f1, f2=55.88; //Две переменные вещественного типа
```

В Java переменным сразу при объявлении задаются стандартные значения (ноль для числовых переменых, false для логических). Так, в приведенном примере в f1 будет 0, а в f2 - 55.88.

Для переменных числового типа определены стандартные арифметические операции: +, -, *, /. Назначение их понятно - все, как в других языках. Как и в других С-подобных языках, есть остаток от деления и операции увеличения/уменьшения на один (%, ++, --). Вот пример:

```
static int k=21, w=10;
...
int z=k%w;
System.out.println(z); //z=1
k++;
System.out.println(k); //k=22
w--;
System.out.println(w); //w=9
```

Операторы ++ и -- можно писать как после, так и до переменной. Чаще всего это все равно, но иногда порядок важен. Вот пример:

```
static int k=10, w=10;
...
int z=k++;
System.out.println(z); //z=10, k=11
z=++n;
System.out.println(z); //z=11, n=11
```

Т. е. у двух форм оператора ++ (это относится и к --) разное возвращаемое значение - в одном случае первоначальное число, а в другом - измененное.

Часть 5. Логический тип

Переменные логического типа могут принимать значение или true (истина), или false (ложь). Сразу обратите внимание, что нельзя вместо true и false писать нулевое и ненулевое значения (как, например, это можно делать в C/C++). Такое замены в Java нет.

Для переменных логического типа существуют операции & (и), && (и), | (или), || (или), ! (не), ^ (исключающее или). Обратите внимание, что для "и" и "или" существует два варианта. Об их различии чуть ниже. Пока же приведем таблицы истинности:

Оператор	Описание	Результат
&& или &	и	Результат true только тогда, когда оба операнда равны true
|| или |	или	Результат false только тогда, когда оба операнда равны false
^	исключающее или	Результат true только тогда, когда ровно один из операндов равен true
!	не	Изменяет значение на противоположное (true на false, false на true)
==	логическое равно	Применяется к перемеым любого типа. Результат true, если оба операнда равны true, false - в противном случае.

Теперь о различии между && и & (|| и |). Операторы & и | - это обычные логические операторы, && и || - сокращенные. Вот конкретный пример для && и &:

```
...
k=20;
if (k<0 & k/0>1) //деление на 0 и возникнет ошибка
{
    System.out.println(k);
}
if (k<0 && k/0>1) //деление на 0, но ошибки нет
{
    System.out.println(k);
}
...
```

Т. е. в первом случае (с одним &) проверяются все части логического выражения, а во втором (с двумя &&) правая часть не проверяется (так как левая равна false).

Вот пример и для "или":

```
...
k=20;
if (k>0 | k/0>1) //деление на 0 и возникнет ошибка
{
    System.out.println(k);
}
if (k>0 || k/0>1) //деление на 0, но ошибки нет
{
    System.out.println(k);
}
...
```

Здесь во втором случае так как левая часть равна true, то правая часть вообще проверяться не будет (и ошибка, соответственно не возникнет).

Также обратите внимание, что в качестве логического равно (т. е. когда мы отвечаем на вопрос, верно ли что что-то равно чему-то) используется двойное равно (==). Одинарное же равно используется для присваивания:

```
if(a==b){ //Если a равно b
{
    a=c; //В a записываем c
}
```

Часть 6. Операторы ветвления

Опрераторов ветления в Java два - if и switch. Первый позволяет пойти программе по одному из двух направлений, второй позволяет сделать выбор между большим числом вариантов (два, три, четыре, ...).

Вот пример программы, которая выводит результат деления одного числа на другое. Если знаменатель равен нулю, то деления не происходит.

```
        int a=30, b=5;
        System.out.println("a="+a);
        System.out.println("b="+b);

        if (b!=0)
        {
            System.out.println(a/b);
        }
        else
        {
            System.out.println("На ноль делить нельзя!!!!");
        }
```

Веточка else не обязательна. Если после проверки условия должен выполниться только один оператор, то фигурные скобки писать не обязательно.

Теперь пример с оператором switch. Пример смотрит, что за символ хранится в переменной ch (+, -, * или /), и в зависимости от этого делает то или иное действие с двумя числами. Результат действия выподится на экран.

```
    char ch='/';
    int k=40, n=10;
    switch(ch)
    {
        case '+':
            System.out.println(k+n);
            break;
        case '-':
            System.out.println(k-n);
            break;
        case '*':
            System.out.println(k*n);
            break;
        case '/':
            System.out.println(k/n);
            break;
        default:
            System.out.println("Error!");
    }
```

Обратите внимание на break. Без него выполнялись бы операторы и в следующем case (пока не встретится break). Например, если написать так:

```
...
case 1:
case 2:
  //Некоторые операторы
  ...
```

то "Некоторые операторы" будут выполнятся и когда проверяемая переменная в switch'e равна 1, и когда она равна 2.

Ветка default будет выполнятся тогда, когда переменная в switch'e не равна ни одному значению в case'ах. Ее использование не обязательно. Если по задаче она не нужно, то не пишите ее.

Часть 7. Операторы циклов

Циклов в Java три вида - while, do-while и for. Первые два следует использовать тогда, когда точно неизвестно, сколько раз цикл должен выполнится. Цикл for используем тогда, когда число, которое наш код должен повторяться, известно.

Вот пример на цикл while:

```
    int n=46;
    int k=0;
    while (k*k<=n)
    {
        k++;
    }
```

В этом примере ищется такое минимальное k, что его квадрат больше n.

Цикл while и его брат цикл do-while выполняются до тех пор, пока условие в круглых скобочках истинно. Как только оно становится равным false, выполнение цикла прекращается.

Пример цикла do-while:

```
    double k;
    do
    {
        k=Math.random();
    }while(k>0.1);
    System.out.println(k);
```

В этом примере ищется первое случайное число меньшее 0.1. Как только оно сгенерировано датчиком случайных чисел Math.random(), выполение цикла прекращается. Math.random() выдает случайные числа от 0 до 1.

Основное отличие циклов while и do-while в том, что while может вообще ни разу не выполнится (если условие в круглыс скобочках сразу false), а do-while выполнится по крайней мере 1 раз.

Заметьте, что для циклов while и do-while где-то внутри цикла обязательно должна меняться переменная, стоящая в круглых скобочках после while. Иначе цикл может стать бесконечным.

И, наконец, пример цикла for:

```
    for (int i=0; i<10; i++)
    {
        System.out.println(i*i);
    }
```

Этот цикл распечатает квадраты целых чисел от 0 до 9. Обратите внимание, что переменная i объявлена прямо внутри цикла. Так часто и делается, так как чаще всего переменная-счетчик цикла вне его не нужна и не используется.

Часть 8. Массивы

Начнем сразу с примеров. Вот пример, в котором мы заводим массив из 3-х целых чисел, в каждое из которых мы записываем случайное целое число от 0 до 9 и затем выводим все числа на экран:

```
class Test{
    public static void main(String [] args)
    {
        int [] k;
        k=new int [3];
        for(int i=0; i<3; i++)
        {
            k[i]=(int)(10*Math.random());
            System.out.println(k[i]);
        }
    }
}
```

В этм примере мы завели массив k за два этапа - сначала мы объявили переменную типа массив целых чисел:

```
...
```

```
        int [] k;
...
```

и затем мы определили массив (т. е. указали, сколько конкретно элементов в нем содержится):

```
...
        k=new int [3];
...
```

Объявление мессива можно сделать и другим способом:

```
...
        int k[];
...
```

Т. е. квадратные скобочки можно ставить и так, и так. В общем-то особой разницы нет, и скобки можно ставить там, где удобнее в конкретной ситуации. Вот небольшой пример на два эти способа:

```
...
        int n, k[]; //n - целое, k - массив из целых
        int[] n, k; //n и k - массивы из целых
...
```

Два этапа для массива можно объединить:

```
...
        int [] k = new int [3];
...
```

Нумерация элементов массива, как и в других C/C++-подобных языках, начинается с нуля.

Задание начальных элементов для массива можно совместить с его объявлением:

```
...
        double[] d = {3.14, -44.43, 9.084};
...
```

В этом примере мы завели массив их трех вещественных чисел. Обратите внимание, что в этом случае можно в квадратных скобках ничего не писать.

Теперь рассмотрим двумерные массивы. Опять же начнем с примера:

```
class Test{
    public static void main(String [] args)
    {
        int [][] k = {{3, 4, -44}, {-2, 8}};
        System.out.println(k[0][2]);
        System.out.println(k[1][1]);
    }
}
```

В этом фрагменте мы заводим двумерный массив (так как у на две пару квадратных скобок) и тут же его инициализируем. Наш массив фактически представляет из себя массив массивов, т. е. в нашем массиве в качестве элементов содержаться два других одномерных массива - в одном из них три элемента, и в другом - два. Обратите внимание, что в Java массивы не обязательно "прямоугольные".

Точно также, как и для одномерных массивов, для многомерных существуют и другие способы их задания:

```
        int [][] k;
        k=new int[2][];
        k[0]=new int[]{3, 4, -44};
        k[1]=new int[]{-2, 8};
```

Также эквивалентны следующие объявления массивов:

```
...
        double[][] d;
...
...
        double[] d[];
...
...
        double d[][];
...
```

Массивы могут участвовать в операциях присваивания:

```
class Test{
    public static void main(String [] args)
    {
        int [][] k = {{3, 4, -44}, {-2, 8}};
        int[][] k1;
        k1=k; //k1 теперь ссылается на точ же массив, что и k
        System.out.println(k1[0][2]);
        System.out.println(k1[1][1]);
    }
}
```

Указанный фрагмент выведет на экран -44 и 8.

Массивы в Java задаются динамически. Это в частности означает, что мы можем менять их размеры в процессе работы программы:

```
        int [] k={3, 4};
        System.out.println(k[1]);
        k=new int[]{2, 6, -55};
        System.out.println(k[2]);
```

В указанном фраменте в массиве k сначала 2 элемента, а затем - 3.

Часть 9. Начинаем изучать классы

С классом мы уже столкнулись при написании нашей первой программы на Java в Частье 3. И это не случайно - без классов на Java нельзя обойтись даже в самой простой программе. На этом же занятии и на нескольких следующих мы с вами и будем изучать классы и все, что с ними связано - наследование, конструкторы, виртуальные функции и другие мудреные вещи.

Давайте создадим новую программу. Вот ее текст:

```
class worker
{
    private int Age;
    public String Name;
    public void setAge(int newAge)
    {
        if(newAge>=0)
            Age=newAge;
        else
            Age=0;
    }
    public int getAge()
    {
        return Age;
    }
}
class Test{
    public static void main(String [] args){
        worker wrk=new worker();
        wrk.setAge(23);
        wrk.Name="Ivan";
        System.out.println(wrk.getAge() + "\n" + wrk.Name);
    }
}
```

Сохраните эту программу в файле Test.java и откомпилируйте. При запуске наша программа должны выдать две строчки: 23 и Ivan.

Что мы в нашей программе делаем? Сначала мы определяем класс worker. Делается это с помощью ключевого слова class:

```
class worker
{
    ...
}
```

В классе мы определяем две переменные - Age для возраста и Name для имени. Кроме типа переменых (int и String) мы используем еще модификаторы доступа - private (означает, что наша переменная не будет видна снаружи класса) и public (с наружи класса доступ есть). Раз переменную Age мы объявили как private, то пишем два метода в нашем классе: setAge для чтения возраста и getAge - для записи. Эти методы мы объявляем с модификатором public, это значит, что мы сможем их вызывать снаружи класса. Метод getAge просто возвращет наш возраст, а метод setAge делает небольшую проверку, и записывает в Age только положительный возраст или нуль в противном случае. Если вы раньше программировали на C++, то обратите внимание, во-первых, что модификаторы доступа ставятся перед каждой переменной и перед каждым методом и во-вторых, что после заакрывающей фигурной скобки класса точку с запятой ставить не надо.

Класс Test служит для испытания класса worker. В нем мы заводим экземпляр нашего класса:

```
    ...
    worker wrk=new worker();
    ...
```

Это мы делаем за два этапа - сначала заводим переменную типа worker (которая является ссылкой на объект), и затем определяем сам объект (с помощью оператора new).

После создания объекта мы можем вызывать его методы, обращаться к открытым переменным и т. п. Это мы и делаем в строчках

```
        ...
        wrk.setAge(23);
        wrk.Name="Ivan";
        System.out.println(wrk.getAge() + "\n" + wrk.Name);
        ...
```

С этим Частьом все!

Часть 10. Наследование классов

Классы можно писать не с нуля, а взяв за основу существующий класс. В этом случае эти классы будут являтся потомком и предком друг для друга. Потомка еще называют подклассом (subclass), а предка - суперклассом (superclass). Еще одна пара названий для таких классов - это класс-потомок и класс-предок.

Если один класс есть потомок другого, то он автоматически умеет делать все то, что умеет делать класс-предок. И нам остаеться только добавить в него то, чего не было в предке или изменить те методы, работа которых в классе-предке нас не удовлетворяет. В этом одна из главных черт Объектно-ориентированного программирования - если у некоторых объектов есть много общего, то можно для них создать класс-предок, в который записать все общие черты. Отличительные же черты будут реализованы в классах-потомках.

Давайте создадим класс boss, который будет потомком для класса worker из прошлого Частья. Вот текст, который введите с том же файле Test.java, в котором находится класс worker:

```
class worker
{
    ...
}
class boss extends worker
{
    public int NumOfWorkers; //Количество подчиненных
```

```
}
```

Ключевое слово extends означает, что наш новый класс boss есть потомок класса worker. Мы добавили в него только переменную NumOfWorkers, в которой будет храниться количество подчиненных. Класс же Test измените следующим образом:

```
    ...
    public static void main(String [] args){
        boss bigBoss=new boss();
        bigBoss.setAge(41);
        bigBoss.Name="Ivan Ivanov";
        bigBoss.NumOfWorkers=100;
        System.out.println(bigBoss.NumOfWorkers + "\n" + bigBoss.Name);
        ...
```

Как вы видите, кроме количества подчиненных для нашего bigBoss мы можем задавать имя и возраст. Они берутся из родительского класса. Все работает, как и ожидалось.

Часть 11. Конструкторы классов

Конструкторы предназначены для задания ннекоторых стандартных значений для переменных-членов класса. Конструктор - это тот же метод класса, только обладающий некоторым количеством особенностей. Раз это метод, значит мы должны после его имени писать круглые скобки, в которых мы можем писать или не писать параметры и т. п. Но есть и несколько черт, отличающих конструктор от других методов класса. Вот они:

Конструктор всегда называется так же, как и класс (т. е. если класс, скажем, называется worker, то и контруктор будет называться worker).

Конструктор в отличие от других методов вызывается сам в момент создания экземпляра класса.

Конструктор не возвращает никакого значения. Это значит, что если перед любым другим методом мы пишем тип возвращаемого значения (int, float и т. п.), то перед конструктором ничего писать не надо (void тоже писать не надо).

Приведем пример конструкторов для класса worker:

```
class worker
{
    private int Age;
    public String Name;
    //Конструктор без параметров
    public worker()
    {
        Age=20;
    }
    //Конструктор с параметрами
    public worker(int newAge, String newName)
    {
        Age=newAge;
        Name=newName;
    }
    ...
}
```

Посмотреть на действия конструкторов в тестовом классе можно так:

```
        ...
        worker wrk1=new worker();
        worker wrk2=new worker(40, "Petrov");
        System.out.println(wrk1.getAge() + "\n" + wrk1.Name);
        System.out.println(wrk2.getAge() + "\n" + wrk2.Name);
        ...
```

Для первого работника вызовется конструктор без параметров и его возраст установится в 20, для второго - конструктор с параметрами, и его возраст станет равным 40, а имя - Petrov

Если вы не заведете в классе конструктора, то компилятор java создаст его сам. Это, кстати, видно в примере Частьа 9:

```
worker wrk=new worker();
```

Здесь worker() - это как раз вызов конструктора без параметров. Создаваемый по умолчанию конструктор ничего не делает.

Часть 12. Абстрактные методы

Когда мы строим иерархию классов, то может оказаться, что некоторый метод должен присутствовать во всех классах, но во всех классах он имеет разную реализацию. Например, у нас может быть много типов работников (директора, программисты, бухгалтера, ...), и все классы для них будет потомками базового класса worker (так как у каждого из них есть возраст, имя, телефон и т. п. и дублировать все это во всех классах смысла нет). Но некоторые методы должны работать по-разному. Например, это может быть метод setSalary() для начисления зарплаты. Для работников с почасовой системой оплаты способ будет один, для совместителей - другой, для постоянных работников - третий.

Так вот, если метод должен присутствовать во всех классах-потомках, и в каждом из них он имеет свою реализацию, то такой метод объявляют в родительском классе абстрактным. Это означает, что в родительском классе мы не пишем реализацию этого метода (и не ставим после него фигурных скобок вообще). Вот пример:

```
abstract class worker
{
    protected int Age;
    public String Name;
    abstract public void setSalary(int newSalary);
    ...
}
class boss extends worker
{
    //Пишем реализацию абстрактного метода родительского класса
    public void setSalary(int newSalary)
    {
        if(newSalary>=0)
            Salary=newSalary;
        else
            Salary=3000;
    }
    ...
}
class engineer extends worker
{
    //Пишем реализацию абстрактного метода родительского класса
    public void setSalary(int newSalary)
    {
        if(newAge>=0)
            Salary=newSalary;
        else
            Salary=2500;
    }
    ...
}
```

Тут мы объявили класс worker, написали в нем абстрактный метод setSalary, и объявили два класса-потомка класса worker - классы boss и engineer. В каждым из их мы пишем свою реализацию для метода setSalary.

Обратите внимание, что если мы объявили некий метод класса абстрактным, то и весь класс надо объявить абстрактным:

```
abstract class worker
    ...
```

Экземпляры абстрактного классы мы создавать не сможем - только экземпляры его потомков.

Для чего вообще возиться с абстрактными методами? Не проще ли его в классе предке вообще не объявлять? Дело в том, что вы можете в переменную типа родительского класса записать экземпляр класса потомка:

```
worker bigBoss=new boss();
```

Это значит, что вы можете в вашем программе объявить массив worker'ов, и записать в него всех работников - и инженеров, и директоров и простых рабтников. И тогда, если вам надо начислить зарплату для всех, то вы просто перебираете элементы этого массива.

Часть 13. Модификатор final

Иногда мы не хотим, чтобы от некоторого созданного нами класса можно было производить классы-потомки. Например, это может понадобиться в целях безопасности.

В этом случае мы объявляем класс с ключевым словом final. Вот пример:

```
final class someclass
{
    //некоторые поля и методы класса
    private int somedata;
    ...
}
```

Теперь от нашего класса someclass нельзя делать классы-потомки. Т. е тут, например, будет ошибка:

```
class newclass extends someclass //Ошибка!
{
    ...
}
```

Часть 14. Статические члены класса

Переменные, которые вы объявляете внутри класса, относятся к определенному экземпляру класса. Например, у класса worker, который мы рассматривали на прошлых Частьах, есть переменная Age (возраст). Понятно, что у одного работника возраст один, а у другого - другой, и эти два возраста между собой никак не связаны. С другой стороны, иногда нам нужна общая переменная на все экземпляры класса. Например, это может быть счетчик количества экземпляров класса. В этом случае такую переменную надо объявить с модификатором static. Вот пример:

```
class someclass
{
    //Счетчик
    static public int number;
    //Конструктор
    public someclass()
    {
        number++; //Увеличиваем счетчик
    }
}
class Test{
    public static void main(String [] args){
        //Создаем два экземпляра класса someclass
        someclass z1=new someclass();
        someclass z2=new someclass();
        System.out.println(someclass.number + "\n");
    }
}
```

Как вы видите, переменная number объявлена как static. Это означает, что она одна на все экземпляры. В конструкторе она увеличивается на 1. Т. е. для первого экземпляра класса она будет равна 1, для второго - 2 и т. д. Что мы и проверяем в классе Test - заводим два экземпляра класса someclass, а затем выводим значение number. Естественно, что number будет равно 2. Обратите внимание, что в строке

```
        ...
```

```
        System.out.println(someclass.number + "\n");
```

переменную number мы извлекаем не из конкретного экземплярра класса (z1, z2), а из самого класса someclass. Это возможно именно потому, что переменная number - статическая. Но, в принципе, в этом месте мы могли бы использовать и конкретный экземпляр класса - результат был бы тот же самый.

Часть 15. Первая оконная программа

Все программы, которые мы создавали до сих пор, были консольными. На этом занятии мы с вами создадим первую программу, которая будет будет иметь оконный вид. Ее можно будет запустить под Windows, и она будет выглядеть как обычная Windows-программа. Конечно, ее можно будет запускать не только под Windows, но и под любой ОС с установленно виртуальной машиной Java. Выглядеть она будет примерно одинаково во всех этих случаях.

Вот текст нашей первой оконной программы. Наберите его в любом текстовом редакторе.

```
import java.awt.*;
class First extends Frame{
    public static void main(String[] args){
        Frame fr=new First();
        fr.setSize(400, 150);
        fr.setVisible(true);
    }
}
```

Сохраните текст в файле с именем First.java.

Небольшой комментарий для написанного нами кода. Для запуска нашей программы в отдельном окне мы создаем (естественно ;)) отдельный класс. У нас он назван First (также, кстати, называется и файл, в котором мы пишем этот текст). Для того, чтобы наше приложение могло работать в отдельном окне, мы объявляем наш класс потомком класса Frame. Для того, чтобы можно было использовать класс Frame, мы в начале программы пишем

```
import java.awt.*;
...
```

В классе мы заводим статический метод main, в котором мы создаем новый экземпляр класса Frame с помощью конструктора First(). Это, собственно, и будет окно нашей программы. Как вы видете, переменная fr имеет тип родительского класса (Frame), а записываем мы в нее экземпляр дочернего класса (First). Далее мы вызываем методы setSize (для установки начальных размеров) и setVisible (для показа окна на экране).

Запустите программу. Если вы работаете из командной строки, то наберите в ней java First и нажмите Enter. Если в некотором java-редакторе, то нажмите соответствующую комбинацию клавиш или кнопку.

Вот результат выполнения программы:

Не пытайтесь закрыть это окно. Все равно это у вас не получится - ни Alt+F4 или крестик в правом верхнем углу не работают.

Часть 16. Читаем строку с клавиатуры

Вы, наверное, обратили внимание, что в предыдущих Частьах мы старательно избегали получать данные от пользователя. Значения всем переменным мы задавали непосредственно в программе. Вот на этом Частье мы и узнаем, как же прочиталь данные, введенные пользователем. Не все пока будет ясно (например, исключения), но код будет работать, и его можно применять в программах.

Вот пример класса, который умеет читать строку с клавиатуры и выводить ее на экран:

```
import java.io.*;
class HelloWorld{
    public static void main(String [] args) throws IOException{
        String s;
        BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
        s=in.readLine(); //Читаем с клавиатуры
        System.out.println("s="+s);
    }
}
```

Что мы тут делаем? Сначалы мы подключаем java.io.*. Это сделано для тогго, чтобы не писать длинные имена. Можно было обойтись и без этого - вот так:

```
...
public static void main(String [] args) throws java.io.IOException{
        ...
        java.io.BufferedReader in=new java.io.BufferedReader(
        new java.io.InputStreamReader(System.in));
...
```

Но так слишком длинно, поэтому первую строчку и добавили.

Далее мы должны завести переменную для буферизованного ввода. Т. е. для вывода мы специальной переменной не заводили, а для ввода должны завести:

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

Переменная наша называется in.

Ну а потом совсем просто - методом readLine мы читаем с клавиатуры, и затем выводим на консоль (экран).

Часть 17. Читаем числа с клавиатуры

Вот пример класса, который может читать числа с клавиатуры:

```
import java.io.*;
class NumReader{
    public static void main(String [] args) throws IOException{
        String s;
        BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
        s=in.readLine(); //Читаем с клавиатуры
        System.out.println("s= "+s);
        double d=Double.valueOf(s).doubleValue(); //Превращение строки в double
        d++;
        System.out.println("d= "+d);
    }
}
```

Для показа, что мы прочитали именно число, мы его сначала увеличиваем на 1, и только после этого выводим на экран.

Если же нам надо превратить строку в целое, то мы должны использовать такую конструкцию:

```
...
    int d=Integer.valueOf(s).intValue();
...
```

Если внимательно присмотреться, то можно увидеть, что мы тут используем странности - Integer вместо int и Double вместо double. Но именно так и должно быть. Дело в том, что это - классы. Класс целых чисел и класс вещественных. Этим классам мы посвятим отдельный Часть.

Часть 18. Закрытие оконной программы

Давайте дополним код нашей первой оконной программы из Частьа 15 по крайней мере таким образом, чтобы она закрывалась:

```
import java.awt.*;
import java.awt.event.*;
class First extends Frame{
    public static void main(String[] args){
        Frame fr=new First();
        fr.setSize(400, 150);
        fr.setVisible(true);
        fr.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent ev)
            {
                System.exit(0);
            }
        });
    }
}
```

Немного пояснений по коду. Так как наша программа должна уметь реагировать на внешние события, то мы добавляем строку

```
...
import java.awt.event.*;
...
```

Далее мы вносим добавления в наш класс First. А именно для созданного внутри него экземпляра fr мы вызываем метод addWindowListener, который добавляет к ннашему классу возможность "прослушивать" оконные события. В качестве параметра метода addWindowListener мы создаем безымянный экземпляр класса WindowAdapter, внутри которого мы пишем обработчик для закрытия окна:

```
    ...
            public void windowClosing(WindowEvent ev)
            {
                System.exit(0);
            }
    ...
```

Понятно, что этот обработчик делает не что иное, как закрывает наше окно (конкретно это делает строка System.exit(0);).

Все! Компилируем и запускаем программу. Теперь наше окно стало еще больше походить на настоящее - его можно закрыть.

Часть 19. Библиотека классов Java - обзор

Классы Java объединяются в пакеты. В число основных пакетов Java входят следующие:

- java.awt - классы и интерфейсы для создания пользовательского интерфейса
- java.applet - классы для создания java-апплетов
- java.io - классы для ввода-вывода (в том числе с консолью и файлами)
- java.net - классы и интерфейсы для работы с сетью
- java.math - классы для различных математических операций ()
- java.lang - содержит наиболее общие для java классы и интерфейсы
- java.util - пакет различных полезных классов (случайные числа, списки, даты и др.)
- java.beans - пакет для создания компонентов JavaBeans
- java.sql - пакет классов и интерфейсов для работы с базами данных
- java.secuirity - пакет классов и интерфейсов для обеспечения безопасности

Более подробно перечисленные пакеты мы рассотрим на последующих Частьах.

Часть 20. Пакет java.awt - обзор

Пакет java.awt предназначен для создания пользовательского интерфейса. Он, в частности, содержит классы для различных компонентов - кнопок, текстовых полей, классы для меню, классы для событий и др.

Вот основные классы этого пакета:

- Button - кнопка
- TextField - текстовое поле
- Checkbox - флажок
- CheckboxGroup - набор радиокнопок
- Label - метка
- List - список
- TextArea - поле для ввода многострочного текста
- Choice - выпадающий список
- Image - абстрактный базовый класс для изображений
- Frame - отображение окна операционной системы
- Component - базовый класс для компонентов
- Container - базовый класс для контейнеров
- Panel - простой контейнер
- BorderLayout - контейнер для компонентов
- Dialog - базовый класс для диалоговых окон
- FileDialog - класс для стандартного диалога открытия или сохранеия файла
- Events - класс для работы с событиями
- AWTEvent - класс для событий AWT
- Font - класс для работы со шрифтами
- Color - класс для работы с цветами (в том числе в формате RGB)
- MenuComponent - абстрактный базовый класс для всех классов меню
- Menu - меню
- MenuBar - меню на линейке меню
- MenuItem - элемент меню
- PopupMenu - контекстное меню
- Cursor - курсор мыши

Часть 21. Библиотека Swing - делаем окно

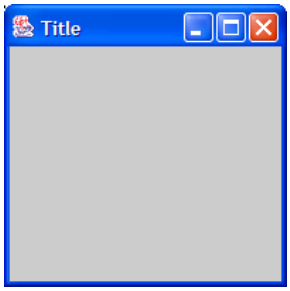
В Java существует две библиотеки для создания пользовательского интерфейса - Awt и Swing. Swing считается более продвинутой и современной. На сегодняшнем Частье мы создадим окно с использованием этой библиотеки.

Итак, приступаем. Вот исходный текст нашей программмы:

```
package progs;
// Импортируем нужные пространства имен.
import javax.swing.*;
// Класс основного окна программы.
public class MyFrame
    extends JFrame {
  // Конструктор.
  public MyFrame(){
    // Устанавливаем размеры и расположение.
    setLocation(400, 200);
    setSize(200, 200);
    // Задаем заголовок окна.
    setTitle("Title");
  }

  public static void main(String[] args) {
    // Создание главного окна.
    new MyFrame().setVisible(true);
  }
}
```

Текст достаточно стандартен (создаем класс, производный от JFrame, после чего создаем его экземпляр в вызове статического метода main) и ясен из комментариев.

А вот так будет выглядеть наша программа после компиляции:



Отметьте, что окно будет закрываться при нажатии на крестик в правом верхнем углу - никакого кода нам для этого не пришлось писать.

Часть 22. Swing: добавляем кнопку с обработчиком

На этом Частье мы посмотрим, как можно добавить к созданному на прошлом Частье окну кнопку с обработчиком.

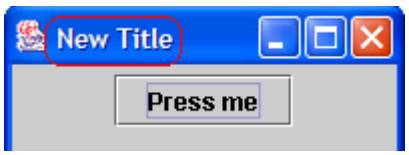
Наша кнопка будет экземпляром класса JButton. Вообще схема работы с элементами управления такая - сначала мы создаем контейнер для различных элементов управления, а потом созданные элементы добавляем к этому контейнеру.

Вот полный текст нашей программы:

```
package progs;
// Импортируем нужные пространства имен.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
// Класс основного окна программы.
public class MyFrame
    extends JFrame {
  // Переменная для кнопки.
  public JButton button;
  // Конструктор.
  public MyFrame(){
    // Устанавливаем размеры и расположение.
    setLocation(400, 200);
    setSize(200, 200);
    setTitle("Title");
    // Задаем контейнер для компонентов.
    Container con = getContentPane();
    con.setLayout(new FlowLayout());

    // Создание кнопки.
    button = new JButton("Press me");
    // Добавление кнопки к контейнеру.
    con.add(button);
    // Добавление обработчика для кнопки.
    button.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
        // Изменение заголовка окна.
        setTitle("New Title");
      }
    });
  }

  public static void main(String[] args) {
    // Создание главного окна.
    new MyFrame().setVisible(true);
  }
}
```

Логика нашей программы такая - создаем в главном окне контейнер, создаем кнопку, добавляем ее к контейнеру, добавляем обработчик для кнопки.

После запуска и компиляции и нажатия на кнопку наша программа будет выглядеть так:



Обратите внимание, что заголовок окна изменился - наш обработчик для кнопки действует!

Часть 23. Swing: кнопка с двумя состояниями

В библиотеке swing наряду с обычной кнопкой есть кнопка с думя состояниями - нажатом и отжатом. Она работает приблизительно как checkbox, только выглядить по-другому. Для создания такой кнопки в приложении мы используем класс JToggleButton.

Вот пример использования такой кнопки:

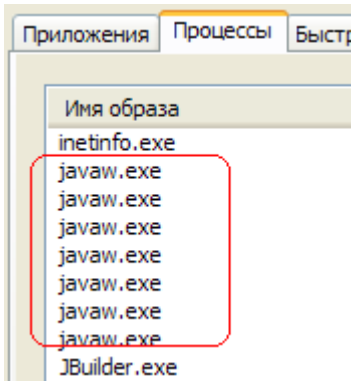
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MyFrame
    extends JFrame {
  // Переменная для кнопки.
  public JToggleButton button;
  // Конструктор.
  public MyFrame(){
    // Устанавливаем размеры и расположение.
    setLocation(400, 200);
    setSize(200, 200);
    setTitle("Title");
    // Задаем контейнер для компонентов.
    Container con = getContentPane();
    con.setLayout(new FlowLayout());

    // Создание кнопки (сразу нажатой).
    button = new JToggleButton("Press me", true);
    // Добавление кнопки к контейнеру.
    con.add(button);
    // Добавление обработчика для кнопки.
    button.addItemListener(new ItemListener(){
      public void itemStateChanged(ItemEvent e){
        // Изменение заголовка окна.
        if(button.isSelected())
        {
          setTitle("Button is selected");
        }
        else
        {
          setTitle("Button isn't selected");
        }

      }
    });
  }

  public static void main(String[] args) {
    // Создание главного окна.
    new MyFrame().setVisible(true);
  }

  public void windowClosing(WindowEvent ev) {
    System.exit(0);
  }
}
```

Часть нашей программы, в которой мы создаем окно, такая же, как и на прошлых Частьах. Отличие только в том, что мы используем класс JToggleButton. С ним мы работаем как всегда - создаем экземпляр, вызываем конструктор (обратите внимание, что в конструкторе мы поставили второй параметр в true, что означает, что кнопка уже нажата), добавляем кнопку к контейнеру и добавляем обработчик для изменения состояния кнопки, в котором мы просто меняем заголовок нашей программы. Обратите внимание, что мы при добавлении этого обработчика указываем itemStateChanged - т. е. нас интересует не столько нажатие на кнопку, сколько изменение ее состояния.

После запуска нашей программы полуаем ожидаемый результат - кнопку с двумя состояниями (при этом в нажатом состоянии кнопка более темная):

Часть 24. Правильное закрытие программы

Созданные нами на прошлых Частьах программы работали не вполне корректно - а именно после своего закрытия они оставались в памяти. Убедится в этом было достаточно просто - запустив и закрыв программу несколько раз, можно было запустить Task Manager (Ctrl + Shift + Escape) и увидеть там несколько запущенных процессов javaw.exe:



Как правильно закрывать программу? Вот как - нам надо явно указать то действие, которое должно выполняться, когда пользователь закрывает программу. Это мы делаем путем вызова метода setDefaultCloseOperation с соответствующим параметром. Например, для правильного закрытия программы и выгрузкой ее из памати надо вызвать этот метод с параметром EXIT_ON_CLOSE (например, в конструкторе класса окна):

```
public class MyFrame
    extends JFrame {
  ...
  public MyFrame(){
    // Задаем действие,
    // выполняемое при выходе из программы.
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    ...
```

Теперь при закрытии программы все будет ОК - в чем не трудно убедиться, запустив Task Manager.

Другие возможные значения параметра для этого метода это DO_NOTHING_ON_CLOSE (ничего не делать), HIDE_ON_CLOSE (просто спрятать программу, не выгружая ее из памяти) и др.

Часть 25. Swing: элемент JCheckBox

Элемент JCheckBox - это обычный checkbox (флажок). Он имеет вид квадратика, в котором может стоять (или не стоять) галочка. Этот элемент может иметь 22 состояния - либо включенное, либо выключенное. При этом состояние каждого флажка не зависит от другох флажков.

Посмотрим, как практически сделать такой элемент.

Вот полный код:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MyFrame
    extends JFrame {
  // Переменная для checkbox'a.
  public JCheckBox checkBox;
  // Конструктор.
  public MyFrame(){
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    // Устанавливаем размеры и расположение.
    setLocation(400, 200);
    setSize(200, 200);
    setTitle("Title");
    // Задаем контейнер для компонентов.
    Container con = getContentPane();
    con.setLayout(new FlowLayout());

    // Создание checkbox'a.
    checkBox = new JCheckBox("Java");
    // Добавление checkbox'a к контейнеру.
    con.add(checkBox);

    // Добавление обработчика для checkbox'a.
    checkBox.addItemListener(new ItemListener(){
      public void itemStateChanged(ItemEvent e){
        // Изменение заголовка окна.
        if(checkBox.isSelected())
        {
          setTitle("CheckBox is selected");
        }
        else
        {
          setTitle("CheckBox isn't selected");
        }
      }
    });
  }

  public static void main(String[] args) {
    // Создание главного окна.
    new MyFrame().setVisible(true);
  }

  public void windowClosing(WindowEvent ev) {
    System.exit(0);
  }
}
```

Самые существенные части этого кода (применительно к данному Частью) следующие:

Во-первых, мы объявляем переменную для нашего checkbox'a:

```
    public JCheckBox checkBox;
```

Во-вторых, мы этот checkbox создаем и присоединяем к нашему контейнеру:

```
    // Создание checkbox'а.
    checkBox = new JCheckBox("Java");
    // Добавление checkbox'а к контейнеру.
    con.add(checkBox);
```

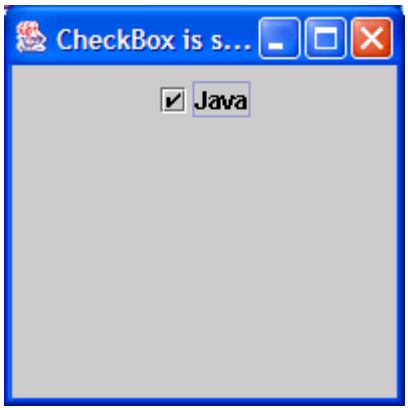
В конструкторе мы сразу указываем ту надпись, которая будет рядом с checkbox'ом.

И в-третьих, мы добавляем обработчик для нашего checkbox'а:

```
    checkBox.addItemListener(new ItemListener(){
      public void itemStateChanged(ItemEvent e){
        // Изменение заголовка окна.
        if(checkBox.isSelected())
        {
          setTitle("CheckBox is selected");
        }
        else
        {
          setTitle("CheckBox isn't selected");
        }
      }
    });
```

В этом обработчике мы просто меняем заголовок окна программы.

Результат выполнения программы будет таким:



Часть 26. Swing: JRadioButton

Элемент управления JRadioButton предназначен для организации выбора только одного значения из нескольких возможных. Для этого несколько элемемнтов JRadioButton объединяются в одну группу, которая работат как единое целое - если выбрать одну из радиокнопок, входящих в группу, то остальные радиокнопки из этой группы становятся невыбранными.

Вот пример кода:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MyFrame
    extends JFrame
    implements ActionListener {
  // Переменные для радиокнопок.
  public JRadioButton r1, r2, r3;
  // Переменная для группы радиокнопок.
  public ButtonGroup bg = new ButtonGroup();
  // Конструктор.
  public MyFrame() {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    // Устанавливаем размеры и расположение.
    setLocation(400, 200);
```

```
    setSize(200, 200);
    setTitle("Title");
    // Задаем контейнер для компонентов.
    Container con = getContentPane();
    con.setLayout(new FlowLayout());

    // Создание радиокнопок.
    r1 = new JRadioButton("Java");
    r2 = new JRadioButton("C/C++");
    r3 = new JRadioButton("C#");
    // Добавление радиокнопок к контейнеру.
    con.add(r1);
    con.add(r2);
    con.add(r3);

    // Добавление радиокнопок в группу.
    bg.add(r1);
    bg.add(r2);
    bg.add(r3);

    // Указание обработчиков для радиокнопок.
    r1.addActionListener(this);
    r2.addActionListener(this);
    r3.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {
    // Изменение заголовка окна в зависимости от выбранной радиокнопки.
    setTitle(e.getActionCommand());
  }

  public static void main(String[] args) {
    // Создание главного окна.
    new MyFrame().setVisible(true);
  }

  public void windowClosing(WindowEvent ev) {
    System.exit(0);
  }
}
```

Обратите внимание на следующе моменты. Во-первых, мы объявили наш класс реализующим интерфейс ActionListener:

```
public class MyFrame
    extends JFrame
    implements ActionListener {
        ...
```

Это мы сделали для возможности добавления обработчика нажатия на наши радиокнопки.

Во-вторых, для объединения радиокнопок в группу мы объявили переменную bg типа ButtonGroup и все радиокнопки в эту группу добавили:

```
    ...
    // Добавление радиокнопок в группу.
    bg.add(r1);
    bg.add(r2);
    bg.add(r3);
    ...
```

В-третьих, мы для всех радиокнопок указали один и тот же обработчик:

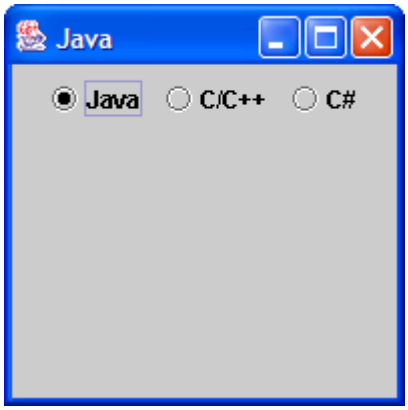
```
    ...
    // Указание обработчиков для радиокнопок.
    r1.addActionListener(this);
    r2.addActionListener(this);
    r3.addActionListener(this);
    ...
```

Этот обработчик - не что иное как метод actionPerformed:

```
  public void actionPerformed(ActionEvent e) {
    // Изменение заголовка окна в зависимости от выбранной радиокнопки.
    setTitle(e.getActionCommand());
  }
```

Этот метод из интерфейса ActionListener, для которого наш класс является потомком и который мы реализуем. Что за кнопку мы выбрали, мы определяем через метод getActionCommand для передаваемого параметра типа ActionEvent.

Запускаем программу. При выборе любой радиокнопки ее текст появится в заголовке окна:



Часть 27. Swing: Несколько групп радиокнопок

На прошлом Частье мы рассмотрели работу с одной группой радиокнопок. Сейчас же мы с вами посмотрим, как работать с несколькими группами радиокнопок.

Приведем сразу листинг программы:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MyFrame
    extends JFrame
    implements ActionListener {
  // Переменные для радиокнопок.
  public JRadioButton r1, r2, r3, r4, r5;
  String sel0, sel1; //Текст выбранной радиокнопки в каждой группе.
  // Две переменные для группы радиокнопок.
  public ButtonGroup bg = new ButtonGroup();
  public ButtonGroup bg1 = new ButtonGroup();
  // Конструктор.
  public MyFrame() {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setLocation(400, 200);
    setSize(200, 200);
    setTitle("Title");

    Container con = getContentPane();
    con.setLayout(new FlowLayout());

    // Создание радиокнопок.
    r1 = new JRadioButton("Java");
    r2 = new JRadioButton("C/C++");
    r3 = new JRadioButton("C#");

    // Добавление радиокнопок к контейнеру.
    con.add(r1);
    con.add(r2);
    con.add(r3);

    // Добавление радиокнопок в группу.
    bg.add(r1);
    bg.add(r2);
```

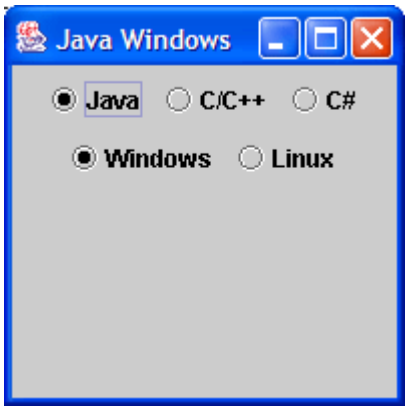
```
    bg.add(r3);

    // Указание обработчиков для радиокнопок.
    r1.addActionListener(this);
    r2.addActionListener(this);
    r3.addActionListener(this);

    // Выделение первой радиокнопки в первой группе.
    r1.setSelected(true);

    // Создание радиокнопок.
    r4 = new JRadioButton("Windows");
    r5 = new JRadioButton("Linux");

    // Добавление радиокнопок к контейнеру.
    con.add(r4);
    con.add(r5);

    // Добавление радиокнопок в группу.
    bg1.add(r4);
    bg1.add(r5);

    // Указание обработчиков для радиокнопок.
    r4.addActionListener(this);
    r5.addActionListener(this);

    // Выделение первой радиокнопки во второй группе.
    r4.setSelected(true);

    // Задание первоначального заголовка окна.
    sel0 = r1.getText();
    sel1 = r4.getText();
    setTitle(sel0 + " " + sel1);
  }

  public void actionPerformed(ActionEvent e) {
    // Изменение заголовка окна в зависимости от выбранной радиокнопки.
    String s = e.getActionCommand();

    // Выясняем изменения в первой группе радиокнопок.
    if(s==r1.getText() || s==r2.getText() || s==r3.getText()){
      sel0 = s;
    }
    // Выясняем изменения во второй группе радиокнопок.
    if(s==r4.getText() || s==r5.getText()){
      sel1 = s;
    }

    setTitle(sel0 + " " + sel1);
  }

  public static void main(String[] args) {
    // Создание главного окна.
    new MyFrame().setVisible(true);
  }

  public void windowClosing(WindowEvent ev) {
    System.exit(0);
  }
}
```

Отличия тут от листинга предыдущего Частья следующие - во-первых, мы добавили еще несколько радиокнопок (r4 и r5), добавили для них группу (bg1). Это мы делали точно также, как и на прошлом Частье. Кроме того, мы сразу в конструкторе выделили первые радиокнопки в каждой группе (через вызов метода setSelected).

Во-вторых, так как у нас две группы, а обработчик для них один, то мы завели в классе две переменные sel0 и sel1 типа String для каждой группы. Каждая из этих переменных просто хранит текст выбранной радиокнопки из каждой группы.

В-третьих, мы в методе actionPerformed выясняем изменения в первой и второй группах радиокнопок и устанавливаем соотвествующий заголовок окна.

Наша программа в работе будет выглядеть так, как и ожидалось - в заголовке окна будут показаны одновременно тексты выбранных радиокнопок из каждой группы:



Часть 28. Swing: Список JList

Класс JList предназначен для выбора пользователем одного или нескольких значений из списка. В этом элементе пользовательского интерфейса можно выбирать как один элемент, так и несколько.

Вот пример применения JList:

```
import javax.swing.*;
public class MyFrame extends JFrame {
    private String [] s = {"red", "yellow", "green"};
    public JList list;
    public MyFrame(){
        JPanel panel = new JPanel();
        list = new JList(s);
        panel.add(list);
        setContentPane(panel);
        setLocation(400, 200);
        setSize(200, 200);
    }
    public static void main(String[] args) {
        new MyFrame().setVisible(true);
    }
}
```

А вот как это будет выглядеть:

Логика приведенного фрагмента достаточно проста: заполняем наш список данными из массива строк (в параметре конструктора), затем добавляем нас список к панели.

Часть 29. Интерфейсы

Интерфейсы в Java предназначены только для объявления некоторых методов и констант. Никакие реализации методов в интерфейсах не предусматриваются. Т. е. интерфейс только содержит

информацию о том, что методы с такими-то названиями в мнем существуют, но не содержит информации, как именно эти методы работают. Реализация же методов интерфейса будет содержаться в классах, которые этот интерфейс реализуют.

Интерфейсы вводятся ключевым словом interface. В объявлении же класса, реализующего некоторый интерфейс, после имени класса идет ключевое слово implements, после которого следует имя интерфейса или интерфейсов (класс может реализовывать несколько интерфейсов).

Вот примеры интерфейса для геометрической фигуры и 3-х классов конкретных герметрических фигур, реализующих этот интерфейс:

```
public interface IFigure {
    double pi = 3.1415;
    // Площадь.
    double square();
}
// Класс прямоугольника.
public class Rect implements IFigure {
    int width = 0;
    int height = 0;
    Rect(int width, int height){
        this.width = width;
        this.height = height;
    }
    public double square(){
        return width * height;
    }
}
// Класс треугольника.
public class Triangle implements IFigure {
    double leg1 = 0;
    double leg2 = 0;
    public Triangle(double leg1, double leg2){
        this.leg1 = leg1;
        this.leg2 = leg2;
    }
    public double square() {
        return leg1 * leg2 * 0.5;
    }
}
// Класс круга.
public class Circle implements IFigure {

    double radius;
    public Circle(double radius){
        this.radius = radius;
    }
    public double square() {
        return radius * radius * pi;
    }
}
```

А вот использование указанного интерфейса и классов, его реализующих:

```
public class Test {
    public static void main(String[] args) {
        Rect r = new Rect(2, 4);
        System.out.println("Rect: " + r.square());

        Triangle t = new Triangle(3, 4);
        System.out.println("Triangle: " + t.square());

        Circle c = new Circle(1);
        System.out.println("Circle: " + c.square());
    }
}
```

Обратите внимание, что каждый класс содержит метод square для вычисления площади. Этот метод обязан быть в каждом классе, так как он есть в интерфейсе, от которвх классы наследуются. Также обратите внимание, что в классе Circle мы использем константу pi из интерфейса. В других языках программирования объявление констант в интерфейсах не допускается.

Выведет программа естественно Rect: 8.0 Triangle: 6.0 и Circle: 3.1415.

Часть 30. Интерфейс в качестве типа

Мы не можем создать экземпляр интерфейса. Именно потому, что у него не существует реализации методов. Т. е. мы не можем написать, например, так:

```
IFigure f = new IFigure(); // Ошибка!
```

С другой стороны в переменную типа интерфейса мы можем записать любой класс, реализующий этот интерфейс. Например, в интерфейс IFigure из прошлого Частья мы можем записать класс Rect:

```
IFigure f = new Rect(2, 4);
```

Это оказывается весьма удобным - так как реальный тип переменной нам может быть не известен на этапе компиляции. Известен же он может быть только на этапе выполнения программы.

Вот так может быть переделан тестовый класс из прошлого Частья:

```
public class Test {
    public static void main(String[] args) {
        IFigure[] f = new IFigure[2];//= //new Rect(2, 4);
        f[0] = new Rect(2, 4);
        f[1] = new Circle(1);
        for (int i = 0; i < 2; i++){
            System.out.println("Square[" + i + "]: " + f[i].square());
        }
    }
}
```

Обратите внимание, что тут мы в переменные типа интерейса (у нас массив из 2-х переменных такого типа) записываем классы-потомки этого интерфейса (а именно Rect и Circle). И в момент вызова метода square интерфейса фактически вызовется метод из соответствующего класса, реализующего этот интерфейс. Т. е. площадь для прямоугольника будет считаться по его формуле, а для круга - по его.

Часть 31. Множественное наследование

В Java, как и во многих других языках программирования, не допускается множественное наследование для классов. У класса может быть только один непосредственный предок (у которого, в свою очередь, может быть и свой один предок и т. п.). Множественное наследование в Java допускается только для интерфейсов. Т. е. в качестве второго (третьего и т. д.) предка может выступать только интерфейс. Комбинации тут возможны разные - например, несколько интерфейсов могут выступать в качестве предков как для класса, так и для интерфейса. Или интерфейсы могут быть предками совместно с одним классом.

Вот несколько примеров (в которых подразумевается, что интерфейсы IInterface1 и IInterface2 существуют):

```
// Интерфейс с 2-я интерфейсами-предками.
public interface IInterface3 extends IInterface1, IInterface2 {
    ...
}
// Класс с 2-я интерфейсами-предками.
public class Class1 implements IInterface2, IInterface1 {
    ...
}
// Класс с интерфейсом и классом в качестве предков.
public class Class2 extends Class1 implements IInterface1 {
    ...
}
```

Обратите внимание, что при наследовании интерфейса от интерфейса и класса от класса мы используем ключевое слово extends, а при наследовании класса от интерфейса мы используем ключевое слово implements.

Часть 32. Сериализация класса

Для того, чтобы экземпляр класса можно было сохранять (например в файл) и потом восстанавливать (например, из файла), класс должен быть сериализуемым. В этом случае сохранение / чтение экземпляра класса будет происходить не поэлементно (что тоже возможно), а целиком.

Вообще же говоря, сериализация - это представление некого объекта в последовательной форме, а десериализация - это восстановление объекта из последовательной формы.

Для того, чтобы класс стал сериализуемым, его надо просто объявить потомком интерфейса Serializable. Методов в этом интерфейсе нет.

Вот сразу пример сериализуемого класса:

```
import java.io.*;
public class Worker implements Serializable {
    // Данные класса.
    int age=0;
    int yoe=0;
    Worker(int age, int yoe){
        this.age = age;
        this.yoe = yoe;
    }

    // Метод main.
    public static void main(String[] args)
throws IOException, ClassNotFoundException {
        // Сериализация экземпляра класса.
        Worker w = new Worker(22, 2);
        ObjectOutputStream os = new ObjectOutputStream(
                new FileOutputStream("1.txt"));
        os.writeObject(w);
        os.close();

        // Десериализация экземпляра класса.
        Worker w1 = new Worker(0, 0);
        ObjectInputStream ois = new ObjectInputStream(
                new FileInputStream("1.txt"));
        w1 = (Worker)ois.readObject();
        ois.close();
        System.out.println("age = " + w1.age + ", yoe = " + w1.yoe);
    }
}
```

Статический метод main, по идее, должен был бы быть в отдельном классе. Мы поместили его в этот же класс для упрощения. В этом методе мы сериализуем в файл 1.txt экземпляр нашего класса (со значениями полей 22 и 2) и потом десериализуем данные из этого файла в другой экземпляр этого же класса Worker. Результат будет вполне ожидаемым - значения полей в новом экземпляре класса будут такие же - 22 и 2.

Часть 33. Инверсия списка

На этом Частье мы рассморим с вами решение классической задачи - инверсии (т. е. размещению элементов в обратном порядке) списка. Список устроен следующим образом: каждый элемент списка содаржит, во-первых, собственные данные и, во-творых, содержить ссылку на следущий элемент списка. Если следующего элемента списка нет, то в ссылке на следующий элемент содержится null. Вот тык выглядит код:

```
public class ListReverse
{
    // Ссылка на следующий элемент списка.
    public ListReverse next = null;
    // Данные.
    public int data = 0;
    // Конструктор.
    public ListReverse(ListReverse next, int data)
```

```
    {
        this.next = next;
        this.data = data;
    }
}
```

Как видно из кода, мы назвали наш класс ListReverse. В нем есть ссылка на следующий экземпляр класса ListReverse, у того на следующий и т. п.. У последнего в списке эта ссылка равна null.

Так как нам надо показывать для проверки содержимое нашего списка до и после инверсии, то напишем метод showAll, который будет показывать все элементы списка, задаваемого своим первым элементом. Для того, чтобы не плодить классы, мы поместим этод метод прямо в наш класс ListReverse, и следаем его статическим. При этом список, подлежащий показу, мы передадим в качестве параметра:

```
    static public void showAll(ListReverse list)
    {
        // Если список пуст.
        if(list == null)
        {
            System.out.println("List is empty");
            return;
        }
        // Пробегаем все лементы списка.
        ListReverse curr = list;
        while (curr != null){
            System.out.println(curr.data);
            curr = curr.next;
        }
    }
```

Этот метод мы пометим прямо в класс ListReverse.

Теперь пишем метод для инверсии. Его мы тоже сделаем статическим методом класса ListReverse, и в качестве параметра в него будет передаваться список для инверсии, а в качестве возвращаемого значения - отинверсированный список. Вот его код:

```
    static public ListReverse reverse(ListReverse list)
    {
        // Если список пуст или содердит только один элемент.
        if(list == null || list.next == null)
        {
            return list;
        }
        // Если в списке больше одного элемента,
        // то заводим вспомогательные переменные
        // и переменную для результата (res).
        ListReverse aux0 = list;
        ListReverse res = list.next;
        ListReverse aux1 = list.next;
        // Этот элемент будет в инвертированном списке последним.
        aux0.next = null;
        // Пробегаем весь список.
        while(aux1.next != null){
            aux1 = aux1.next;
            res.next = aux0;
            aux0 = res;
            res = aux1;
        };
        res.next = aux0;
        return res;
    }
```

С классом ListReverse все. Теперь напишем класс для проверки. Вот его код:

```
public class ListReverseTest {
    public static void main(String[] args) {
        // Создаем список a.
        // Последний элемент списка.
        ListReverse a = new ListReverse(null, 0);
        a = new ListReverse(a, 1);
        a = new ListReverse(a, 22);
```

```
        a = new ListReverse(a, -441);

        // Показ элементов списка до инверсии.
        ListReverse.showAll(a);
        // Инверсия списка.
        a = ListReverse.reverse(a);
        // Показ элементов списка после инверсии.
        ListReverse.showAll(a);
    }
}
```

Результатом программы будет, естественно, -441 22 1 0 0 1 22 -441.