ООП

Структурное программирование, операторы цикла, условные операторы, процедуры, функции, взаимодействующие объекты, созданные на основе классов.

Класс описывает определённый тип, объединяющий не только данные, но и программный код для обработки этих данных.

Под инкапсуляцией подразумевают механизм, который позволяет объединить в объекте данные и код для обработки данных, скрыв и защитив при этом от пользователя внутреннюю реализацию объекта. Взаимодействие пользователя с объектом ограничивается предопределенным набором действий и правил. Эти правила, вместе с механизмом их реализации, формируют интерфейс объекта или его спецификацию. В языке Java инкапсуляция реализуется на уровне концепции классов и объектов.

• Благодаря полиморфизму объекты с одинаковой спецификацией/ интерфейсом могут иметь разное содержимое или реализацию. Девиз полиморфизма можно сформулировать так: "Один интерфейс – много реализаций!". Главным практическим проявлением полиморфизма является возможность создавать методы с одинаковыми именами или даже сигнатурами (тип результата, имя метода, список аргументов), но выполняющихся по-разному. В Java полиморфизм реализуется путем переопределения и перегрузки методов (смысл этих терминов описан в книге) с учетом наследования классов.

• Такой механизм, как наследование, позволяет создавать новые классы на основе уже существующих классов. При этом новый класс автоматически получает (или наследует) свойства исходного (или родительского) класса. Согласно принятой в Java терминологии, родительский класс называется суперклассом, а созданный на его основе класс – подклассом. Кроме унаследованных из суперкласса свойств, в подклассе могут добавляться новые свойства. Благодаря наследованию, формируется своеобразная структура или иерархия классов. В вершине иерархии находятся классы наиболее общего характера, а дальше по цепочке наследования располагаются классы более узкого "профиля". При этом класс-наследник реализует спецификацию родительского класса, в результате чего методы работы с объектами производного класса аналогичны методам работы с объектами родительского класса.

библиотеке графических компонентов, которая называется Swing. Это название присутствует в инструкции javax.swing.\*

Инструкция public static void main(String[] args) называется сигнатурой метода

Согласно общепринятым правилам названия методов для удобства указываются с пустыми круглыми скобками.

Переменная – это именованная ячейка памяти, к которой в программном коде обращаются через имя. (Функцией называется именованный блок программного кода, который можно вызывать в программе по имени.)

Выделяют базовые типы переменных (их еще называют простыми или примитивными) и переменные классовых типов (переменные типа класса - их еще называют ссылочными переменными)

Числовые значения для типа byte лежат в диапазоне от -128 до 127. Значения типа short могут принимать значения от -32768 до 32767. Типом int поддерживаются значения в диапазоне от -2147483648 до 2147483647. Наконец, типом long поддерживаются значения в диапазоне от -9223372036854775808 до 9223372036854775807.

Для типа double памяти для хранения данных (в битах) выделяется в два раза больше, поэтому и диапазон шире. Переменная типа ﬂ oat позволяет работать с числами со значениями (по модулю) до 3,4×1038. Чувствительность переменной (минимальный шаг дискретности) составляет величину порядка 3,4×10-38. Точность значений составляет при этом до 8 цифр в представлении числа. В то же время максимальное значение и минимальный шаг дискретности для переменной типа double составляют соответственно 1,7×10308 и 1,7×10-308. Точность в представлении числа обеспечивается на уровне 17 цифр. Совершенно очевидно, что тип double намного предпочтительней.

Диапазон значений СHAR лежит в пределах от 0 до 65535.

Объявление переменной означает, что для нее в памяти выделяется место. При этом объем выделяемой памяти определяется типом переменной. Например, для переменной типа double это 8 байт (64 бита). Переменная типа ﬂ oat получает в свое распоряжение 4 байта (32 бита). Под переменные целочисленных типов byte, short, int и long выделяется соответственно 1, 2, 4 и 8 байтов (8, 16, 32 и 64 бита). Переменная типа char занимает в памяти 2 байта (16 бит).

Первое присваивание значения переменной называется инициализацией.

ситуация (когда при объявлении переменной инициализация выполняется выражением, содержащим другие переменные) называется динамическим объявлением переменной.

Классы-оболочки используют в тех случаях, когда необходимо реализовать числовое, логическое или символьное значение в виде объекта.

Арифметические операторы: + - / \* %. Они бинарные (2 операнда). Декремент и инкремент ++ -- унарные, префиксные и постфиксные.

Логические операторы: & (амперсанд) - оператор логического И (конъюнкция), | (вертикальная черта) - оператор логического ИЛИ (дизъюнкция), ^ (каре) – оператор исключающего ИЛИ, ! (восклицательный знак) – логическое отрицание. Бинарные все кроме !.

Операторы сравнения: <, <=, >, >=, ==, !=.

Побитовые операторы: &, |, ^, ~, << (оператор сдвига влево), >>> (оператор сдвига вправо без смещения старшего бита), >> (оператор сдвига вправо). Предназначены для выполнения операций с числами на уровне их побитового представления. Результатом являются числа. Бинарные кроме ~, <<, >>>, >>.

Тернарный оператор: a?b:c.. Три операнда. Если а true то b иначе c.

Оператор присваивания: (сюда) **=** (это).

Явное приведение и автоматическое расширение типов.

Byte x=100, y=100;

Byte z=x+y; // Error!!! x+y – int!!!

Byte z=(byte)(x+y); // !!!!!!!!

Типы литералов.

Литералы – это числа, буквы, текст, который мы присваивали в качестве значений переменным.

если нет крайней необходимости, целочисленные переменные объявляются с типом int, а действительные – с типом double.

Сокращённые формы операторов.

Byte z=z+1; // Error !!!!!!

Ho byte: z+=1;

Инкремент и декремент префиксный и постфиксный:

Int a=100;

Int b=2\*++a; // 202

Int b=2\*a++; // 200

Вычисление выражений и приоритет операций:

Слева на право

Следующий список содержит разбивку базовых операторов по приоритетам (от наивысшего приоритета до низшего): 1. Наивысший приоритет имеют круглые скобки ( ), квадратные скобки [ ] и оператор "точка" .. 2. Операторы инкремента ++, декремента --, побитового ~ и логического отрицания !. 3. Оператор умножения \*, деления / и вычисления остатка %. 4. Оператор сложения + и вычитания –. 5. Операторы побитовых сдвигов >>, << и >>>. 6. Операторы сравнения больше >, больше или равно >=, меньше или равно <= и меньше <. 7. Операторы сравнения равно == и неравно !=. 8. Оператор &. 9. Оператор ^. 10. Оператор |. 11. Оператор &&. 12. Оператор ||. 13. Тернарный оператор ?:. 14. Оператор присваивания =, и сокращенные формы операторов: +=, -=, \*=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=.

Управляющие операции.

If else (else if), switch case (break) default, while, do-while, for,

Массивы.

В Java все массивы динамические.

Сам массив представляет собой область памяти, содержащую фактические данные, то есть значения элементов массива. Переменная массива фактически содержит адрес этой области памяти.

Одномерные.

при объявлении переменной одномерного массива указывается тип элементов массива, пустые квадратные скобки и имя переменной. int[] MyArray; (MyArray=new int[10];)

Массив создается независимо от переменной массива. После создания массива переменной массива необходимо в качестве значения присвоить ссылку на массив.

{В процессе работы программа оперирует с данными, которые реализуются в виде простых переменных, массивов, объектов. Для них нужно выделять память. Память можно выделять: а) при компиляции программы (то есть до того, как программа запущена на выполнение); б) непосредственно при выполнении программы. В первом случае говорят о статическом распределении памяти. Если память выделяется в процессе выполнения программы, говорят о динамическом распределении памяти. В Java память для массивов и объектов выделяется непосредственно в процессе выполнения программы, то есть динамически. Именно для динамического выделения памяти служит оператор new. При этом для переменной массива память выделяется статически.}.

командой double[] x,y объявляются две переменные x и y массива из doubleэлементов, а командой double x[],y объявляется переменная массива x и обычная переменная y типа double.

MyArray[0]. MyArray.length.

Присваивание и сравнение массивов.

int[] ﬁrst, second; ﬁrst=new int[10]; second=ﬁrst; // две ссылки на один и тот же объект.

int[] ﬁ rst,second; ﬁ rst=new int[10]; second=new int[5]; second=ﬁ rst; // теряем второй объект

Результатом выражения ﬁ rst==second является true, только если переменные ﬁ rst и second ссылаются на один и тот же массив. Даже если массивы одинаковые, но физически разные, результатом сравнения будет false. Чтобы сравнить два физически разных массива, необходимо выполнять поэлементную проверку.

Двумерные массивы.

двумерный массив – это одномерный массив, элементами которого являются одномерные массивы.

double[][] NewArray=new double[3][4]; { double[] NewArray[] или double NewArray[][].} Первый индекс означает строку в матрице, а второй – столбец,

Инициализация массива.

Размер массива вычисляется автоматически в соответствии с количеством значений, переданных для инициализации. int[] nums={1,3,5,7}; double[][] x={{1,2,3,4},{5,6,7,8},{9,10,11,12}}; char[][] symbs={{'a'},{'b','c'},{'d','e','f'}};

Альтернативные команды объявления массивов int[] nums=new int[]{1,3,5,7}; double[][] x=new double[][]{{1,2,3,4},{5,6,7,8},{9,10,11,12}}; char[][] symbs=new char[][]{{'a'},{'b','c'},{'d','e','f'}}; Результат один и тот же.

Работа с массивами.

Метод пузырька:

**for**(i=0; i<size; i++) {

**for**(j=0; j<size-i-1; j++) {

**if**(symbs[j]>symbs[j+1]) {

s=symbs[j+1];

symbs[j+1]=symbs[j];

symbs[j]=s;

}

}

/\*(i – int) \*/ symbs[i]=(char)(s+(byte)(Math.random()\*26)) // рандом символов

Упрощённая форма перебора массива:

for (тип\_переменной переменная: массив){ команды } // for (int s:a){

Нормально разберись в массивах, дебил!

Классы и объекты.

три базовых принципа ООП: инкапсуляция, полиморфизм и наследование.

Структурирование программы через систему классов и объектов является реализацией механизма инкапсуляции. В основу механизма инкапсуляции положена идея об объединении данных и программного кода для обработки этих данных в отдельные структурные единицы - объекты. Благодаря полиморфизму существенно сокращается количество методов (функций) для обработки данных, поскольку полиморфизм подразумевает использование общих методов для решения однотипных задач и выполнения похожих действий. В Java полиморфизм реализуется через систему перегрузки и переопределения методов. наследование – механизм, который позволяет создавать одни классы на основе других, уже существующих классов.

В классе описывают Поля – это переменные, которые описываются в пределах класса. Методы – это функции, которые описываются в пределах класса. Собственно, все.

объектная переменная – переменная, которая содержит в качестве значения адрес объекта.

Объект создается с помощью команды вида new имя\_класса() // MyClass obj=new MyClass();

обращения к полю объекта: имя объекта, точка и имя поля. Это так называемый точечный синтаксис. // obj.num

Методы.

Метод представляет собой именованный блок программного кода, который выполняется при обращении к методу. В Java методы описываются в классах. Описание метода состоит из сигнатуры (заголовка, или шапки) и тела метода.

Вот пример описания метода: double func(double x){ double t; t=2\*x+1; return t; }

Конструкторы. (метод, который выполняется вначале, при создании объекта.)

• Имя конструктора совпадает с именем класса. • Конструктор не возвращает результат, а в сигнатуре конструктора не указывается идентификатор типа возвращаемого результата – никакой! • У конструктора могут быть аргументы, которые описываются так же, как и аргументы обычного метода. Его так же можно перегружать.

Перегрузка методов и конструкторов.

Это создание нескольких версий одного метода – все версии имеют одно и то же имя, но разные сигнатуры. (сигнатура метода состоит из идентификатора типа результата, имени метода и списка аргументов.)

Присваивание объектов.

Создание нескольких ссылок на один и тот же объект // objA=objB

Объекты, на которые в программе нет ни одной ссылки, автоматически отслеживаются и удаляются из памяти.

Создание копий объекта.

Конструктор класса (два аргумента): MakeObjCopy(String str,int num){ name=str; number=num;}

Конструктор создания копии: MakeObjCopy(MakeObjCopy obj){ name=obj.name; number=obj.number;} // новый объект

Метод для создания копии объекта: MakeObjCopy copy(){

Создание объекта-копии: MakeObjCopy tmp=new MakeObjCopy(name,number);

Возвращение результата: return tmp;} // новый объект

Статистические поля и методы.

статическое поле или метод являются общими для всех экземпляров класса, то есть для всех объектов этого класса. Как следствие, к статическим полям или методам можно обращаться безотносительно к конкретному объекту. Более того, даже если ни один объект для класса не создан, статические поля и методы все же доступны.

Обращаем внимание, что при обращении к статическому полю обращение выполняется через имя класса, то есть в формате имя\_класса.статическое\_поле.

Обращаем внимание читателя, что при обращении к статическому методу используется имя класса (команда вида MyMath.sin(аргументы)), а объект для класса MyMath вообще не создается.

Объекты и методы.

В Java есть два способа передачи аргументов: по значению и по ссылке. При передаче аргументов по значению, на самом деле в метод для обработки передаются не сами аргументы, а их технические копии. При передаче аргументов по ссылке операции выполняются непосредственно с аргументами.

Массивы и объекты.

Как и в случае с объектами, в массиве следует различать переменную массива (переменную, которая в качестве значения содержит ссылку на массив) и непосредственно сам массив. массивы и объекты могут использоваться "совместно": как массивы объектов, как поля-объекты и как поля-массивы, например.

Анонимные объекты.

При создании анонимных объектов команда создания объекта есть (инструкция с оператором new), а команды записи (присваивания) ссылки на объект в объектную переменную нет. Несмотря на это, из таких объектов можно, например, вызывать методы или указывать анонимный объект аргументом метода.

Внутренние классы.

Один класс может описываться внутри другого класса. Класс, который описывается внутри другого класса, называется внутренним. Класс, в котором описывается внутренний класс, называется внешним, или классом-контейнером. Главная особенность внутреннего класса состоит в том, что в нем доступны поля и методы внешнего класса. Вместе с тем, поля и методы внутреннего класса не доступны во внешнем. Недоступны в том смысле, что к ним нельзя обратиться напрямую. Однако при этом можно создать объект внутреннего класса и из него вызвать метод внутреннего класса или обратиться к полю внутреннего класса. //

Аргументы командной строки.

//

Наследование.

class имя\_подкласса extends имя\_суперкласса{ код\_подкласса; }

Фактически, с некоторыми ограничениями, речь идет о том, что код суперкласса автоматически неявно каптируется в программный код подкласса. И хотя поля и методы суперкласса в подклассе не описаны, они там есть и к ним можно обращаться и использовать, как если бы они были явно описаны в подклассе.

Конструктор подкласса.

При создании конструктора подкласса необходимо предусмотреть способ вызова конструктора суперкласса. Для этого в код конструктора подкласса включается инструкция super с круглыми скобками, в которых перечисляются аргументы, которые передаются конструктору суперкласса.

Переопределение методов.

Суть переопределения методов связана с тем, что программный код унаследованного в подклассе метода может быть переопределен. В результате подкласс имеет такой же метод (с таким же названием), что и суперкласс, но выполняются они по-разному.

названия полей подкласса могут дублировать названия полей из суперкласса. Например, если в суперклассе A описано поле int number, а в подклассе B, который создается на основе класса A, также явно описано поле int number, то по умолчанию, если в классе B выполнить ссылку number, получим доступ к тому полю, что описано в этом подклассе. Чтобы получить доступ к полю, описанному в суперклассе, используем ссылку super.number.

Наличие ключевого слова ﬁnal в сигнатуре метода запрещает его переопределение. Наличие ключевого слова ﬁnal в сигнатуре класса запрещает на его основе создавать подклассы. (Наличие ключевого слова ﬁnal в описании переменной означает, что значение этой переменной не может быть изменено. Такая переменная фактически является константой, и значение ей должно быть присвоено при объявлении.)

члены (поля и методы), объявленные в суперклассе с ключевым словом private, в подклассе не наследуются.

Разница между закрытыми и открытыми членами проявляется, только если мы пытаемся получить доступ к членам класса извне, то есть из другого класса.

Объектные переменные суперклассов.

Объектные переменные (// для себя: обычно это obj) суперклассов могут ссылаться на объекты подклассов к тем полям и методам подкласса, которые описаны в суперклассе. объектная переменная суперкласса позволяет обращаться к полям и методам подкласса, которые унаследованы из суперкласса. В случае методов, если имеет место их переопределение, то вызывается версия метода из подкласса.

Абстрактные классы и интерфейсы.

Класс может содержать абстрактные методы. Абстрактный метод, в отличие от обычного, имеет сигнатуру, но не содержит блока с программным кодом. Абстрактный метод объявляется с ключевым словом abstract. После сигнатуры абстрактного метода ставится точка с запятой. Класс, содержащий хотя бы один абстрактный метод, также называется абстрактным. Описывается такой класс с использованием все того же ключевого слова abstract. Для абстрактного класса не может быть создан экземпляр класса, то есть объект.

рекурсия - в коде описания метода вызывается этот же метод (но с другим аргументом). int F(int n){ if(n==1) return 1; else return n\*F(n-1);}

Интерфейсы // не разобраны

Если неабстрактный класс реализует несколько интерфейсов, то в нем должны быть описаны методы из каждого реализуемого интерфейса. Каждый метод описывается с ключевым словом public.

Пакеты и уровни доступа.

Файл может содержать только одну package-инструкцию, и она должна быть первой строкой в файле. Если такой инструкции нет вовсе, то описанные в классе классы и интерфейсы по умолчанию относятся к безымянному пакету. Ссылка на другой пакет import mypack.subpack.MyClass;

1. В файле может быть только один открытый класс, то есть класс, описанный с ключевым словом public.

2. Если в файле есть открытый класс, то имя файла должно совпадать с именем этого класса.

3. Если в файле есть package-инструкция, то она указывается первой строкой кода.

4. В файле может быть несколько import-инструкций. Все они размещаются в начале файла, но после package-инструкции (если такая есть).

Чтобы запомнить, где какой член доступен, можно воспользоваться несколькими простыми правилами.

1. Члены класса, описанные с ключевым словом public, доступны везде, в том числе и во внешних классах других пакетов.

2. Члены, описанные без ключевого слова уровня доступа, доступны везде в пределах пакета, но не доступны за его пределами, вне зависимости от того, о внешнем классе или подклассе идет речь.

3. Член класса, описанный с ключевым словом private, доступен только в классе, где он описан.

4. Член класса, описанный с ключевым словом protected, доступен везде, кроме случая, когда речь идет о внешнем классе из внешнего пакета.

Работа с текстом и другие утилиты.

Также важно понимать разницу между сравнением строк, например, с помощью метода equals() и оператора ==. Кратко разница такая. Методом equals() сравнивается текстовое содержимое разных объектов. Значение true возвращается, если текст в объектах одинаков. При этом сами объекты физически различны. Оператором == значение true возвращается, если соответствующие объектные переменные ссылаются на один и тот же объект.

Помимо класса String, для работы с текстом могут использоваться, например, классы StringBuffer и StringTokenizer. Класс StringBuffer принципиально отличается от класса String тем, что объекты, созданные на основе класса StringBuffer, могут изменяться. Другими словами, если текст реализован с помощью объекта класса StringBuffer, то редакторские правки в текст могут вноситься прямо в этом объекте. Достигается это за счет того, что при создании объектов класса StringBuffer выделяется дополнительное место, чтобы в значение объекта можно было вносить изменения. По умолчанию при создании объекта класса StringBuffer, кроме фактически текста, выделяется место еще для 16-ти символов. Именно за счет этого дополнительного места редактирование текста выполняется прямо в объекте.

Cмотреть стр. 214

Исключительные ситуации их типы.

Класс Throwable подклассы Exception и Error. Error – фатальные ошибки. RuntemeException (подкласс Exception).

Неконтролируемые исключения:

ArithmeticException

Ошибка, связанная с выполнением арифметических операций (например, деление на ноль на множестве целых чисел)

ArrayIndexOutOfBoundsException

Ошибка возникает в том случае, если индекс массива выходит за допустимые границы

ArrayStoreException

Ошибка возникает при присваивании элементу массива значения несовместимого типа

ClassCastException

Ошибка возникает при попытке выполнить приведение несовместимых типов IllegalArgumentException

Ошибка возникает, если методу указан неправильный аргумент

IllegalMonitorStateException

Ошибка возникает при работе с монитором (относится к многопоточному программированию) IllegalStateException

Ошибка возникает, если ресурс находится в некорректном состоянии

IllegalThreadStateException

Ошибка возникает, если предпринята попытка выполнить некорректную операцию на потоке IndexOutOfBoundsException

Ошибка связана с тем, что индекс выходит за допустимый диапазон значений NegativeArraySizeException

Ошибка возникает при попытке создать массив отрицательного размера

NullPointerException

Ошибка возникает при некорректном использовании ссылок (обычно когда объектная переменная содержит пустую ссылку)

NumberFormatException

Ошибка, которая возникает при преобразовании строки к числовому значению (когда в число преобразуется строка, содержащая некорректное текстовое представление числа) SecurityException Ошибка возникает при попытке нарушить режим защиты

StringIndexOutOfBoundsException Ошибка индексирования при работе с текстовой строкой UnsupportedOperationException Ошибка возникает при попытке выполнить некорректную операцию

Контролируемые исключения:

ClassNotFoundException Ошибка связана с тем, что невозможно найти нужный класс CloneNotSupportedException Некорректная попытка клонировать объект

IllegalAccessException Ошибка доступа к ресурсу

InstantiationException Ошибка, связанная с попыткой создать объект абстрактного класса или интерфейса

InterruptedException Ошибка, связанная с прерыванием одного потока другим NoSuchFieldException Ошибка связана с тем, что отсутствует нужное поле

NoSuchMethodException Ошибка связана с тем, что отсутствует нужный метод

Если метод может выбрасывать (генерировать) контролируемое исключение, и обработка этого исключения в методе не предусмотрена, то в сигнатуре этого метода необходимо указать ключевое слово throws и имя класса контролируемого исключения.

try{

// код, который может вызвать исключение

}catch(класс\_исключения объект\_исключения)

{ // код для обработки исключения

}

??? при переборе ошибок и типов параметров в catchблоках считается и ситуация, когда класс исключения является подклассом класса объекта, указанного параметром catch-блока. Поэтому, например, чтобы отслеживать и обрабатывать ошибки всех типов, можно для catchблока указать параметр класса Exception.

// разобраться с примером стр. 233. (как вывести «всё в порядке»?)

Многопоточное программирование. // thread (нить) не stream (поток)

Части программы, которые выполняются параллельно, называются потоками, а сам подход, базирующийся на использовании потоков, называется многопоточным программированием.

Класс Thread c методами:

activeCount() Метод в качестве результата возвращает количество активных потоков в текущей группе потоков

currentThread() Методом в качестве результата возвращается ссылка на текущий объект потока (ссылка на текущий поток)

getId() Методом в качестве результата возвращается идентификатор потока - уникальное целое число, генерируемое при создании потока

getName() Метод в качестве результата возвращает имя потока, из которого вызывается этот метод

getPriority() Методом в качестве результата возвращает приоритет потока

getThreadGroup() Методом в качестве значения возвращается группа, к которой принадлежит текущий поток

holdsLock() Метод в качестве значения возвращает логическое значение. Значение равно true, если метод удерживает монитор объекта, переданного в качестве аргумента методу. В противном случае возвращается значение false. Монитор - объект, который используется для блокировки ресурса потоком и недопущения одновременного обращения к ресурсу разных потоков. Концепция монитора используется для синхронизации работы потоков

interrupt() Методом выполняется прерывание потока

isAlive() Метод используется для того, чтобы определить, используется ли поток

join() Метод-инструкция ожидания завершения потока

run() Метод определяет точку входа в поток. Программный код потока задается путем переопределения этого метода

setName() Методом задается имя потока (имя потока указывается в виде текстовой строки аргументом метода)

setPriority() Методом задается приоритет потока (приоритет передается аргументом методу)

sleep() Метод используется для приостановки выполнения потока. Время (в миллисекундах), на которое выполняется приостановка в работе потока, указывается аргументом метода

start() Метод для запуска потока. При вызове этого метода автоматически запускается на выполнение метод run(), объявленный в интерфейсе Runnable

wait() Метод переводит поток в режим ожидания

yield() Метод временно приостанавливает текущий поток и позволяет выполнение других потоков

по умолчанию определяет приоритет потока (метод main() главный, от него по цепочке подпотоки «дочерние»).

создание дочернего потока:

Выполнение потока — это выполнение метода run(), который объявлен в интерфейсе Runnable (причем это единственный метод, объявленный в интерфейсе). То есть, чтобы создать поток, как минимум где-то нужно описать метод run(). Удобно это сделать в классе, который реализует интерфейс Runnable. Класс Thread реализует интерфейс Runnable. Однако в классе Thread метод run() имеет пустую реализацию (то есть описан без рабочего кода, с пустым телом). Поэтому напрямую воспользоваться классом Thread не получится, как и обойтись без него — запускается поток методом start(), который вызывается из объекта класса Thread. Поэтому у нас вырисовываются следующие перспективы относительно создания нового потока: 1. Реализовать в классе пользователя интерфейс Runnable, описав в классе метод run(). Это тот самый код, который выполняется в рамках нового потока. В этом классе также необходимо предусмотреть создание объекта класса Thread и вызов из этого объекта метода start(). Обычно все это реализуется в конструкторе класса. 2. Сделать все точно то же самое, но создать класс пользователя не расширением интерфейса Runnable, а наследуя класс Thread. В этом случае в качестве объекта, из которого вызывается метод start(), может использоваться экземпляр класса пользователя.

команда thA.join() означает, что главный метод (команда вызывается из главного метода) будет ожидать завершения потока thA.

Чтобы выполнить синхронизацию метода (в этом случае доступ к нему может иметь только один поток), в сигнатуре метода указывается ключевое слово synchronized. Чтобы выполнить синхронизацию ресурса (объекта), используем такой шаблон: synchronized(объект\_синхронизации){ код\_синхронизации }

Система ввода/вывода.

В Java есть класс System, у которого три статических поля in, out и err, которые являются потоками данных (потоковыми переменными) соответственно ввода, вывода и потока ошибок.

System.out называется потоком стандартного вывода,

System.err называется потоком ошибок,

System.in называется стандартным потоком ввода.

создать объект класса Scanner и связать этот объект со стандартным потоком ввода System.in. Процедура банальная: при вызове конструктора класса Scanner объект стандартного ввода System.in указывается аргументом конструктора.

hasNext()

Методом возвращается логическое значение true, если в строке ввода еще есть слова, и false в противном случае

hasNextInt()

Методом возвращается логическое значение true, если в строке ввода еще есть int-значение, и false в противном случае

hasNextLong()

Методом возвращается логическое значение true, если в строке ввода еще есть long-значение, и false в противном случае

hasNextDouble()

Методом возвращается логическое значение true, если в строке ввода еще есть double-значение, и false в противном случае

hasNextBoolean()

Методом возвращается логическое значение true, если в строке ввода еще есть логическое значение, и false в противном случае

hasNextByte()

Методом возвращается логическое значение true, если в строке ввода еще есть byte-значение, и false в противном случае

hasNextShort()

Методом возвращается логическое значение true, если в строке ввода еще есть short-значение, и false в противном случае hasNextLine() Методом возвращается логическое значение true, при наличии строки ввода, и false в противном случае hasNextFloat() Методом возвращается логическое значение true, если в строке ввода еще есть ﬂ oat-значение, и false в противном случае next() Методом возвращается следующее слово в строке ввода nextInt() Методом возвращается следующее int-значение в строке ввода nextLong() Методом возвращается следующее long-значение в строке ввода nextDouble() Методом возвращается следующее double-значение в строке ввода nextBoolean() Методом возвращается следующее логическое значение в строке ввода nextByte() Методом возвращается следующее byte-значение в строке ввода nextShort() Методом возвращается следующее short-значение в строке ввода nextLine() Методом возвращается текстовая строка ввода nextFloat() Методом возвращается следующее ﬂ oat-значение в строке ввода

Scanner input=**new** Scanner(System.***in***);

name=input.nextLine();

Форматированный вывод.

Под форматированным выводом будем подразумевать явное определение параметров представления данных в консольном окне — таких, например, как количество цифр в дробной части действительных чисел, наличие или отсутствие знака у положительных чисел, параметры данных типа дата/время и многое другое.

Основу форматированного консольного вывода составляет метод printf(),

d Целое (десятичное) число

x Целое шестнадцатеричное число

o Целое восьмеричное число

f Действительное число с плавающей (или фиксированной) точкой

e Действительное число с плавающей точкой в экспоненциальном формате

g Число с плавающей точкой в общем формате

a Действительное число с плавающей точкой в шестнадцатеричном представлении

s Текстовая строка

c Символ

b Логическое значение

h Хэш-код

% Символ процента. Разделитель блоков в строке форматирования

+ Инструкция выводить знак для положительных чисел

пробел Инструкция добавлять пробел перед положительными числами

0 Инструкция выводить ведущие нули

( Инструкция заключать отрицательные числа в скобки

, Использование системного разделителя групп (тысячных разрядов)

# Инструкция для действительных чисел всегда добавлять десятичную точку. Для шестнадцатеричных и восьмеричных чисел означает добавление префикса 0x и 0 соответственно

$ Инструкция используется для индексирования аргументов в строке форматирования. Индекс указывается перед символом $

< Инструкция используется в качестве ссылки на аргумент, который обрабатывался в предыдущем блоке форматирования

Общий шаблон блока текстовой строки, определяющей формат вывода аргумента, имеет следующий вид: %[индекс\_аргумента$][индикатор][размер\_поля][.знаков\_после\_ запятой]символ\_типа\_данных

Работа с файлами.

Для связывания потоков ввода и вывода данных с файлами используются классы FileInputStream и FileOutputStream. Эти классы используются для создания байтовых потоков ввода и записи данных в файлы. Чтобы открыть нужный файл для чтения или записи, необходимо создать объект класса FileInputStream (чтение из файла) и FileOutputStream (запись данных в файл). Аргументом конструктора класса указывается имя открываемого файла. Конструкторы обоих классов могут выбрасывать необрабатываемое исключение класса FileNotFoundException. После завершения работы с файлом его необходимо закрыть close().

// разобрать стр. 269 – 270,,, s -1?

int s=ﬁn.read() Если считана инструкция окончания файла, значением переменной s будет -1

Основы библиотеки Swing

Библиотеки графических утилит: Abstract Window Toolkit AWT (тяжёлые), Swing (лёгкие)

компоненты и контейнеры. От контейнера компонент отличается тем, что может содержать (и, как правило, содержит) другие компоненты и контейнеры. Компонент, в свою очередь, для отображения на экране должен быть включен в состав контейнера. Поэтому, по крайней мере, один контейнер при создании графического интерфейса придется использовать. Если контейнеров в программе несколько, то один всегда является контейнером верхнего уровня, в котором содержатся все прочие компоненты и контейнеры..

Классы, не содержащие начальной литеры J в названии, относятся к библиотеке AWT, то есть соответствуют "тяжелым" компонентам.

Класс Описание

JApplet Класс предназначен для работы с апплетами

JButton Класс предназначен для создания кнопок

JCheckBox Класс для создания опций

JCheckBoxMenuItem Класс для создания пункта меню в виде опции

JColorChooser Класс для создания палитры выбора цвета

JComboBox Класс для создания раскрывающихся списков

JComponent Суперкласс для классов компонентов

JDesktopPane Контейнер, используемый для создания многодокументного интерфейса

JDialog Контейнер верхнего уровня для создания диалоговых окон

JEditorPane Класс для создания панели редактирования с поддержкой стилей

JFileChooser Класс для создания диалогового окна выбора файлов

JFormattedTextField Класс для создания поля редактирования с поддержкой форматирования

JFrame Класс для создания окна

JInternalFrame Класс для создания внешнего окна, которое может содержать внутренние подокна

JLabel Класс для создания меток

JLayeredPane Класс для управления механизмом перекрытия компонентов

JList Класс для создания списка

JMenu Класс для создания стандартного меню

JMenuBar Класс для создания меню верхнего уровня

JMenuItem Класс для реализации пункта меню

JOptionPane Класс для работы с основными типами диалоговых окон

JPanel Класс панели

JPasswordField Класс для создания поля ввода пароля

JPopupMenu Класс для создания контекстного меню

JProgressBar Класс для создания индикатора хода процесса

JRadioButton Класс для создания переключателя опции

JRadioButtonMenuItem Класс для создания меню в виде переключателя опций

JRootPane Класс корневой панели

JScrollBar Класс для создания полосы прокрутки

JScrollPane Класс для создания панели с полосой прокрутки

JSeparator Класс для создания разделителя в меню

JSlider Класс для создания линейного регулятора

JSpinner Класс для создания инкрементного регулятора (список с кнопками со стрелками, предназначенными для изменения значения в списке)

JSplitPane Класс для создания специальной панели, состоящей из двух частей с подвижным разделителем

JTabbedPane Класс для создания панели с вкладками

JTable Класс для создания таблиц

JTextArea Класс для создания текстовой области

JTextField Класс для создания поля редактирования

JTextPane Класс текстовой панели широкого спектра

JToggleButton Класс для создания кнопки с двумя состояниями

JToolBar Класс для создания панели инструментов

JToolTip Класс предназначен для реализации оперативных подсказок для элементов интерфейса

JTree Класс для реализации древовидной иерархической структуры данных

JViewport Класс реализации канала просмотра

JWindow Контейнер для реализации окон (без строки названия и системных пиктограмм)

В Java используется модель обработки событий с делегированием. Реализуется она по следующей схеме. В компонентах, с которыми что-то может происходить, регистрируются специальные обработчики событий.

Классы событий

Классы событий пакета java.awt.event

ActionEvent Событие происходит при нажатии кнопки, двойном щелчке на элементе списка или выборе пункта меню

AdjustmentEvent Событие происходит при выполнении операций с полосой прокрутки

FocusEvent Событие происходит при получении или потере компонентом фокуса

ItemEvent Событие происходит, если выбран или отмечен элемент списка или меню

KeyEvent Событие генерируется при выполнении ввода с клавиатуры

MouseEvent Событие связано с перемещением мыши, нажатием и отпусканием клавиш мыши, наведением курсора в область компонента и выводом его из области

MouseWheelEvent Событие связано с вращением колесика мыши

WindowEvent Событие происходит при активизации окна, закрытии, деактивации, сворачивании и разворачивании и прочее

ComponentEvent Событие происходит при изменении компонента, скрытии, отображении, изменении размера и прочее

ContainerEvent Событие происходит при добавлении (удалении) компонента в контейнер (из контейнера)

InputEvent Абстрактный суперкласс для классов, описывающих события, связанные с вводом для компонентов

TextEvent Событие происходит при изменении значения текстового поля или текстовой области

Классы событий пакета javax.swing.event

AncestorEvent Событие связано с добавлением, удалением или перемещением объекта-предка компонента

CaretEvent Событие связано с изменением позиции курсора в текстовом компоненте

ChangeEvent Событие происходит при изменении состояния компонента

HyperlinkEvent Событие связано с операциями с гиперссылкой

ListDataEvent Событие происходит при изменении содержимого списка

ListSelectionEvent Событие происходит при выборе (отмене выбора) элемента списка

MenuEvent Событие генерируется при выборе или отмене выбора пункта меню

TableModelEvent Событие связано с изменением модели таблицы TreeExpansionEvent Событие связано с разворачиванием или сворачиванием дерева TreeModelEvent Событие связано с изменением модели дерева

TreeSelectionEvent Событие происходит при выборе узла дерева

// addСобытиеListener()

Классы-адаптеры и соответствующие им интерфейсы

Класс-адаптер Интерфейс, реализуемый в классе FocusAdapter FocusListener

KeyAdapter KeyListener

MouseAdapter MouseListener

MouseMotionAdapter MouseMotionListener

MouseInputAdapter MouseListener и MouseMotionListener

WindowAdapter WindowListener

ComponentAdapter ComponentListener

ContainerAdapter ContainerListener

Апплеты.

Апплеты выполняются под управлением браузера. Апплет - небольшая программа, которая в откомпилированном виде размещается на сервере, импортируется через Internet и выполняется как составная часть веб-документа. Апплеты имеют ограниченный доступ к системным ресурсам, поэтому представляют собой относительно безопасный инструмент программирования. //java.applet

полная инструкция подключения апплета в htmlдокументе может выглядеть следующим образом: <applet code="имя\_апплета" width=размер\_ширина height=размер\_ высота> </applet>

Резюме

Глава 1

1. Любая программа на Java содержит хотя бы один класс. В этом хотя бы одном классе должен быть метод с названием main(), который называется главным методом программы. Выполнение программы отождествляется с выполнением этого метода.
2. Если для ввода или вывода информации используются диалоговые окна, необходимо подключить библиотеку Swing, для чего в начале программы размещается import-инструкция. В этой библиотеке есть класс JOptionPane, через который реализуются окна для ввода и вывода сообщений.
3. Программы, которые мы в ближайшее время будем писать, имеют следующий шаблон (его следует запомнить):import javax.swing.\*; public class HelloWorld{ public static void main(String[] args){ // здесь основной код } }
4. В программе (пока что программа для нас - метод main() и содержащий его класс) могут и обычно используются переменные. Переменная – это ячейка памяти, которая имеет имя и в которую можно записывать значения и считывать значение из нее. Обращение к переменной в программе осуществляется по имени.
5. Перед использованием переменной в программе ее нужно объявить. При объявлении переменной указывается ее тип и имя. Мы пока знаем только один тип переменной – текстовый String.
6. Переменной присваивается значение. Оператором присваивания в Java служит знак равенства =. Переменная, которой присваивается значение, указывается слева от оператора присваивания. Присваиваемое значение указывается справа от оператора присваивания.

Глава 2

1. В Java переменные имеют тип. Тип переменной определяет, какие данные могут записываться в переменную, какой объем памяти выделяется под переменную и какие операции могут выполняться с переменной.
2. В Java существует несколько базовых, или простых, типов данных. Это целочисленные типы (byte, short, int и long), действительные числовые типы (ﬂ oat и double), символьный тип char (значение – буква или символ) и логический тип boolean (два значения – true и false).
3. При объявлении переменной указывается тип ее значения и имя. Одновременно с объявлением можно переменной присвоить значение (инициализировать переменную). Несколько переменных можно объявлять одновременно. Объявление и инициализация переменной производится в любом месте метода, но до ее первого использования.
4. Константа от переменной отличается тем, что значение константы после инициализации (которая выполняется вместе с объявлением) изменить нельзя. Константы описываются так же, как переменные, но для них указывается еще ключевое слово ﬁ nal.
5. Для базовых типов существуют классы-оболочки: Byte, Short, Integer, Long, Float, Double, Character и Boolean соответственно для типов byte, short, int, long, ﬂ oat, double, char и boolean. Класс-оболочка предназначен для реализации того или иного значения в виде объекта. У классов-оболочек, кроме прочего, имеется ряд полезных методов. Например, у класса Integer есть статический метод parseInt(), с помощью которого текст, содержащий представление целого числа, преобразуется в число

Глава 3

1. В Java существует четыре группы операторов: арифметические, логические, операторы сравнения и побитовые операторы.
2. Оператор деления имеет особенность - если операнды целочисленные, деление выполняется нацело.
3. У операторов инкремента и декремента есть префиксная и постфиксная формы. Для префиксной формы сначала изменяется операнд, а затем вычисляется выражение, частью которого является операнд. Для постфиксной формы сначала вычисляется выражение, а затем изменяется операнд.
4. Тернарный оператор представляет собой упрощенную форму условного оператора.
5. В одном выражении может быть несколько операторов присваивания.
6. Существуют сокращенные формы операторов, которые позволяют оптимизировать синтаксис команды по изменению значения переменной.
7. При несовпадении типов переменных в выражении выполняется автоматическое приведение (расширение) типов. Приведение типов можно выполнять и в явном виде.
8. По умолчанию целочисленные литералы относятся к типу int, а действительные литералы относятся к типу double.
9. Значения выражений вычисляются с учетом приоритета операторов. Для изменения порядка вычисления выражения и повышения читабельности кода рекомендуется использовать круглые скобки.

Глава 4

1. Операторы цикла и условные операторы позволяют создавать точки ветвления в программе и многократно повторять однотипные действия.
2. К условным операторам относят if-оператор и switch-case-оператор (оператор выбора). К операторам цикла относят операторы forоператор, while-оператор и его модификацию do-while-оператор.
3. В условном if-операторе проверяется условие, и если оно истинно, выполняется блок команд после условия. В противном случае выполняются команды после ключевого слова else. Может использоваться упрощенный вариант оператора без else-ветки.
4. В switch-case-операторе вычисляется значение выражения, и после этого в соответствии с полученным значением выполняется тот или иной case-блок.
5. В операторах цикла блок команд выполняется циклически до тех пор, пока остается истинным некоторое условие. В зависимости от типа оператора цикла, условие проверяется перед началом выполнения блока ко манд или после него. Если при вычислении условия получено значение false, выполнение оператора цикла прекращается.
6. Инструкция break используется для завершения работы оператора цикла или оператора выбора. Инструкция continue позволяет досрочно завершить один итерационный цикл.

Глава 5

1. Массивы в Java создаются динамически с помощью оператора new. Ссылка на массив записывается в переменную массива.
2. Для объявления переменной одномерного массива указывают базовый тип элементов, пустые квадратные скобки и имя массива (переменную массива).
3. Для создания одномерного массива после оператора new указывается тип элементов массива и в квадратных скобках - его размер. Оператором new возвращается ссылка на созданный массив. Эта ссылка может быть записана в переменную массива.
4. Индексация элементов массива всегда начинается с нуля.
5. Для доступа к элементу одномерного массива после имени массива в квадратных скобках указывается индекс элемента в массиве.
6. Для определения размера массива используют свойство length.
7. Двумерный массив - это массив, элементами которого являются переменные одномерных массивов.
8. При объявлении двумерных массивов используются две пары квадратных скобок.

Глава 6

1. Класс представляет собой конструкцию, в которой объединены данные и программный код для обработки этих данных. Конкретный экземпляр класса называется объектом.
2. Класс может содержать переменные (поля класса) и методы для обработки полей и не только (методы класса). Поля и методы класса называются членами класса.
3. Описание класса начинается со слова class, затем следует имя класса и в фигурных скобках код класса.
4. Объекты класса создаются динамически с помощью оператора new. После оператора указывается имя класса и в круглых скобках аргументы, передаваемые конструктору. Как результат возвращается ссылка на созданный объект. Эта ссылка обычно записывается в объектную переменную.
5. При объявлении объектной переменной указывают имя класса и имя этой переменной.
6. Конструктором класса называется метод, который вызывается автоматически при создании объекта. Конструктор имеет такое же название, как класс, и для него не указывается тип возвращаемого результата.
7. Методы (и конструкторы) могут перегружаться. В этом случае создается несколько вариантов метода с одним и тем же именем, но разными сигнатурами. Какой вариант метода следует вызывать, определяется исходя из контекста команды с вызовом метода.

Глава 7

1. В классе могут существовать статические поля и методы. Они описываются с ключевым словом static, являются общими для всех объектов класса и существуют независимо от того, сколько создано объектов и созданы ли они вообще.
2. Для обращения к статическому полю или вызова статического объекта указывают имя класса и через точку имя поля или метода.
3. Объекты могут передаваться аргументами методам, могут возвращаться в качестве результата. При этом следует помнить, что объект и переменная объекта - не одно и то же. Объект - это объект. Переменная объекта - ссылка на объект.
4. Существует два механизма передачи аргументов методам: по значению и по ссылке. При передаче аргументов по значению на самом деле в метод передаются их копии. При передаче аргументов по ссылке передаются непосредственно аргументы. Аргументы базовых типов передаются по значению. Объекты передаются по ссылке (за счет использования объектных переменных).
5. Ссылку на объект не обязательно записывать в объектную переменную. В этом случае получаем анонимный объект.
6. Один класс может описываться внутри другого. Внутренний класс имеет доступ ко всем полям и методам класса-контейнера.
7. Если программа запускается с параметрами, то эти параметры передаются аргументом методу main() в качестве текстового массива.

Глава 8

1. Наследование — это механизм, который позволяет одним классам получать свойства других классов (наследовать их).

2. Класс, который создается на основе уже существующего класса, называется подклассом. Исходный класс называется суперклассом.

3. При описании подкласса после его названия через ключевое слово extends указывается имя суперкласса. У подкласса может быть только один суперкласс.

4. Подкласс может быть суперклассом для другого класса.

5. При описании конструктора подкласса необходимо предусмотреть механизм передачи аргументов конструктору суперкласса.

6. Наследуемые из суперкласса методы можно переопределять в подклассе.

7. Объектная переменная суперкласса может ссылаться на объект подкласса. При этом существует доступ только к тем членам подкласса, которые объявлены в суперклассе.

8. Класс может содержать только объявление метода (без его описания). Такой класс (и метод) описывается с ключевым словом abstract и называется абстрактным. У абстрактного класса нет объектов.

9. Интерфейс напоминает абстрактный класс. Интерфейс содержит только объявления методов и статические константы. Описывается с использованием ключевого слова interface.

10. На основе интерфейсов создаются классы. Класс, который реализует интерфейс, содержит в своей сигнатуре ключевое слово implements, после которого указывается имя интерфейса.

11. Один класс может реализовывать сразу несколько интерфейсов.

12. Описываемый в классе метод интерфейса должен содержать publicинструкцию.

13. Один интерфейс может наследовать (расширять) другой интерфейс.

14. Интерфейсная переменная — переменная, в качестве типа которой указано имя интерфейса. Такая переменная может ссылаться на объект класса, реализующего интерфейс. Доступ возможен только к методам класса, объявленным в интерфейсе.

15. В Java классы и интерфейсы группируются по пакетам. Для группировки классов по пакетам используют package-инструкцию. Для подключения пакетов используют import-инструкцию.

Глава 9

1. В Java текст — это, как правило, объект класса String (или StringBuffer).

2. "Текстовые" классы имеют несколько конструкторов и большой арсенал методов для работы с текстом.

3. Переопределяя метод toString(), можно задавать закон преобразования объектов классов пользователя к текстовому формату.

4. В Java имеется набор встроенных классов для работы с датой и временем: Date, Calendar, GregorianCalendar, TimeZone и другие.

5. Полезным при математических вычислениях будет класс Math. В нем описаны методы для основных (элементарных) математических функций.

Глава 10

1. При возникновении ошибки в процессе выполнения программы автоматически создается объект, описывающий эту ошибку, — объект называется исключение, а сложившаяся ситуация — исключительной.

2. В Java существует иерархия классов, предназначенных для обработки исключительных ситуаций.

3. Если исключительную ситуацию не обработать программными методами, она приведет к "аварийному" завершению работы программы.

4. Обработка исключительных ситуаций выполняется в блоке try-catch. В try-блоке размещается контролируемый программный код, а в catchблок помещается код, предназначенный для обработки исключительной ситуации.

5. Помимо использования встроенных классов, для обработки исключительных ситуаций пользователь/программист может описывать собственные классы исключений.

Глава 11.

1. Несколько фрагментов кода программы могут выполняться одновременно (параллельно). В этом случае говорят о потоках выполнения.

2. В Java поддерживается встроенная многопоточная модель.

3. Арсенал средств многопоточного программирования реализуется через класс Thread и интерфейс Runnable.

4. Поток в Java — это объект класса Thread, его подкласса или класса, реализующего интерфейс Runnable.

5. Для получения ссылки на поток используют статический метод currentThread() класса Thread.

6. Для создания нового потока необходимо описать метод run(), объявленный в интерфейсе Runnable (в классе Thread метод run() имеет пустую реализацию). Этот метод определяет набор действий, выполняемых в рамках потока.

7. Для запуска потока из Thread-объекта вызывается метод start().

8. В некоторых случаях необходимо выполнять синхронизацию - регулировать совместный доступ потоков к общим ресурсам.

Глава 12.

1. Консольный вывод в Java может быть реализован на основе стандартного потока вывода — статического поля-объекта out класса System. Кроме стандартного потока вывода System.out, полезными в работе могут быть стандартный поток ввода System.in и поток ошибок System. err.

2. Один из наиболее простых способов реализации консольного ввода подразумевает создание объекта класса Scanner. Этот объект, в свою очередь, создается на основе потока ввода System.in. 3. Для форматированного консольного вывода используем метод printf().

4. Неконсольный ввод/вывод может быть организован средствами класса JOptionPane библиотеки Swing. В этом случае ввод и вывод данных осуществляется с помощью диалоговых окон.

5. Для создания файлового ввода/вывода могут, кроме прочего, использоваться классы FileInputStream и FileOutputStream, FileReader, FileWriter, InputStreamReader, OutputStreamWriter, BufferedReader и BufferedWriter.

Глава 13

1. Приложения с графическим интерфейсом в Java создаются на основе классов библиотеки AWT или Swing.

2. Для основных компонентов графического интерфейса существуют встроенные классы. Создание компонента подразумевает создание объекта соответствующего класса.

3. На основе существующих классов можно создавать собственные классы для компонентов с предопределенными свойствами.

4. В Java используется система обработки событий с делегированием. Для обработки того или иного события создается объект-обработчик. Объект-обработчик регистрируется в компоненте, который может вызвать событие.

5. Классы для обработчиков событий создаются путем реализации стандартных интерфейсов или наследованием специальных классовадаптеров.

Глава 14

Чтобы научиться программировать – надо программировать.

Глава 15

1. Апплет представляет собой программу, которая выполняется под управлением браузера.

2. Создаются апплеты на основе класса Applet библиотеки AWT или класса JApplet библиотеки Swing.

3. У апплетов нет метода main(), роль конструктора играет метод init(). Каждый раз при обращении к странице с апплетом вызывается метод start(). Метод stop() вызывается при уходе со страницы, а метод destroy() — при завершении работы апплета.

4. В html-документ апплет добавляется с помощью пары тегов <applet> и </applet>.

5. Для передачи апплету параметров через html-документ используют тег <param>.

6. В апплете могут использоваться практически те же компоненты графического интерфейса, что и в обычном приложении.

If/else в строку для буленовского выражения : char canDrive = (fuelLevel > 0) ? 'Y' : 'N';

/// факториал (произведение натуральных чисел, например 5!=5⋅4⋅3⋅2⋅1) Допустим, необходимо вычислить выражение F(5). При его вычислении загружается код метода F() и сначала проверяется, равен ли аргумент единице. В данном случае это не так. Поэтому как результат возвращается выражение 5\*F(4). Чтобы вычислить это выражение, снова загружается код метода и рассчитывается значение F(4). Для этого вычисляется выражение 4\*F(3), и так далее, пока не будет вычислено выражение 2\*F(1). Потом эта цепочка сворачивается в обратном направлении. Поэтому следует учесть, что, хотя рекурсия позволяет создавать эффектный программный код, он не всегда эффективный с точки зрения экономии использования системных ресурсов.