Лекция 10

Java. Основные понятия. Комментарии, переменные, типы данных, операторы.

ООП и Java

Вот уже многие годы язык Java входит в число самых популярных и востребованных. Он красивый, эффектный и, самое главное, очень производительный. Но, к сожалению, не самый простой. Именно поэтому спрос на Java-программистов неизменно высок. Язык Java – это бриллиант, который украсит багаж знаний любого программиста.

Концепция ООП базируется на трех фундаментальных принципах: инкапсуляции, полиморфизме и наследовании.

Инкапсуляция подразумевает объединение в одно целое данных и кода для обработки этих данных. Обычно инкапсуляция реализуется через использование классов и объектов. В разных языках программирования все происходит по-разному.

Полиморфизм базируется на использовании универсальных интерфейсов для решения однотипных задач. Этот механизм в Java реализуется за счет перегрузки и переопределения методов.

ООП и Java

Наследование позволяет создавать новые классы на основе уже существующих классов, что значительно экономит усилия, сокращает объем кода и повышает его надежность.

Эти три принципа в той или иной степени присущи любому языку программирования, поддерживающему парадигму ООП.

Наиболее популярными объектно-ориентированными языками программирования сегодня принято считать, наряду с **Java**, языки C++ и C#.

Язык **Java** изначально предназначался для написания больших и сложных программ. **Java** в полной мере поддерживает концепцию ООП, являясь полностью объектно-ориентированным языком. Это означает, что любая программа, написанная на языке **Java**, должна содержать описание по крайней мере одного класса.

Особенности языка Java

Язык **Java** был разработан (с 1991 по 1995 год) инженерами компании **Sun Microsystems**, которую затем поглотила корпорация **Oracle** (**www.oracle.com**). Именно **Oracle** теперь и отвечает за поддержку технологии **Java**. Исторически стимулом к созданию Java (если точнее, то речь шла о разработке целой технологии) стало желание получить программные средства для работы с бытовыми приборами.

Здесь на первый план выходит проблема универсальности программных кодов. И так совпало, что как раз в это время начали бурно развиваться интернет-технологии. Подход, использованный в **Java**, во многом идентичен подходу, примененному при разработке программ, предназначенных для работы во Всемирной паутине.

Именно это обстоятельство и способствовало быстрому росту популярности **Java** (***неформальный девиз Java звучит как «написано однажды – работает везде»***).

Особенности языка Java

Если вы создаете исходный код, заранее зная, на каком компьютере и с какой операционной системой он будет выполняться, то у вас есть возможность оптимизировать программу под параметры исполнительной среды.

Если же вы пишете программу, предназначенную для использования в Интернете, то заранее не знаете ни тип процессора, ни тип операционной системы, которые использует конечный потребитель программного продукта. Особенностью интернет-среды является принципиальное разнообразие используемых операционных систем и аппаратного обеспечения. Отсюда – требование к универсальности кодов.

Проблема универсальности программных кодов решена в **Java** в рамках концепции виртуальной **Java** - машины (**JVM**, от **Java Virtual Machine**). Так, если обычно при компиляции программы (например, написанной на C++) на выходе мы получаем исполнительный машинный код, то в результате компиляции **Java** - программы получают промежуточный байт-код, который выполняется не операционной системой, а виртуальной **Java** -машиной, представляющей собой специальную программу.

Особенности языка Java

Разумеется, предварительно виртуальная **Java** - машина должна быть установлена на компьютер пользователя. Такую виртуальную машину называют **Java Runtime Environment (JRE)** – среда запуска приложений **Java**. С одной стороны, это позволяет создавать достаточно универсальные программы (в том смысле, что они могут использоваться с разными операционными системами). Но с другой стороны, платой за универсальность является снижение скорости выполнения программ. Кроме того, следует четко понимать, что язык **Java** создавался именно для написания больших и сложных программ. Писать на **Java** консольные программы, которые выводят сообщения вроде "**Hello, world**!", – это все равно что на крейсере отправиться на ловлю карасей. Тем не менее **Java** позволяет решать и такие примитивные задачи.

Особенности языка Java

Рассмотрим код:

```
class Demo{ public static void main(String[] args){
    System.out.println("Приступаем к изучению Java!");
    }
}
```

Код размещается между открывающей и закрывающей фигурными скобками. Первая, внешняя, пара нужна для определения кода класса, вторая – для определения метода этого класса.

Создание программы в **Java** начинается с описания класса. Описание класса начинается с ключевого слова **class**. После этого следует уникальное имя класса. В данном случае класс называется **Demo**. Код класса размещается в блоке из фигурных скобок.

Если назвали класс **Demo**, то файл, в котором описывается этот класс, должен называться **Demo.java**. В противном случае программа может не скомпилироваться.

Особенности языка Java

Каждая программа, написанная на **Java**, имеет один и только один ***главный метод***. Этот метод обязательно называется **main**() и описывается по определенным правилам с определенными атрибутами.

Класс, в котором описан главный метод, называется ***главным классом***. Выполнение программы начинается с выполнения кода главного метода. Поэтому команды, которые размещаются в теле главного метода, – это именно те команды, которые выполняются в программе.

Главный метод описывается в соответствии с таким шаблоном:

```
class название_класса {
    public static void main(String[] args){
    // Команды в теле главного метода
   }
 }
```

Помимо того что метод должен называться **main**(), в его описании (перед названием метода) указываются ключевые слова **public**, **static** и **void**. Ключевое слово **public** означает, что метод является открытым и он доступен за пределами класса, в котором описан.

Особенности языка Java

Методы и поля, описываемые в классе, могут иметь разный уровень доступа. Уровень доступа определяется специальным ключевым словом (**`public`**, **`private`** или **`protected`**) или его отсутствием.

Ключевое слово **`static`** означает, что метод статический и для его вызова нет необходимости создавать объект класса.

Методы бывают статические и обычные (нестатические). Статический метод, хотя он и описывается в классе, существует сам по себе. Чтобы вызвать такой метод, достаточно знать, в каком классе он описан. Нестатический метод спрятан в объекте. Чтобы вызвать такой метод, сначала нужно создать объект. ***Главный метод должен быть статическим!***

Ключевое слово **`void`** означает, что метод не возвращает результат.

Особенности языка Java

В круглых скобках после имени главного метода указана инструкция **`String[] args`**. Она описывает аргумент **`args`** главного метода. Аргументом является текстовый массив (набор значений текстового типа - текстовые значения в **`Java`** реализуются как объекты класса **`String`**).

Аргумент главного метода может понадобиться в случае, если при вызове программы ей передаются параметры. Тогда эти параметры считываются (в текстовом формате) и из них формируется массив. Ссылка на этот массив передается в главный метод через аргумент. Если параметры программе не передаются (а они обычно не передаются), то аргумент главного метода играет чисто декоративную роль. Но описать его следует в любом случае. Поэтому инструкция **`String[] args`** в описании главного метода тоже является обязательной.

Особенности языка Java

Команда **`System.out.println("Приступаем к изучению Java!")`**. Команда заканчивается точкой с запятой — это стандарт для **`Java`** (все команды заканчиваются точкой с запятой). Командой с помощью встроенного метода **`println`**() в окне вывода печатается сообщение "**`Приступаем к изучению Java`**!". Текст сообщения указан аргументом метода. Команда вызова метода дает нам пример использования классического точечного синтаксиса. Итак, есть библиотечный класс **`System`**. У этого класса есть статическое поле **`out`**, которое связано с потоком вывода. Поле **`out`** является объектом. И у этого объекта имеется метод **`println`**(), который нам нужно вызвать. В таком случае мы указываем полную иерархию: класс, поле, метод.

Разделителем является точка. То есть инструкция вида **`System.out.println`**() означает, что нужно в классе **`System`** найти объект **`out`** и вызвать из него метод **`println`**().

Комментарии в Java

В **Java** существует три типа комментариев:

- однострочный комментарий

  // однострочный комментарий.

- многострочный комментарий.

  / многострочный*
  комментарий/*

- многострочный комментарий документирования

  */** комментарий*/*

(Начинается последовательностью символов /** и заканчивается последовательностью символов */. Обычно используется в случае, если планируется автоматическое генерирование документации).

Переменные в Java

Переменная – это идентификатор (имя), с помощью которого можно получить доступ к памяти для записи значения или считывания значения.

Переменную удобно представлять в виде ячейки с названием (имя переменной), и в эту ячейку можно что-то положить или посмотреть, что там находится.

Данные, которые в принципе можно хранить с помощью переменных, разбиты на типы: например, целые числа, действительные числа, символы, логические значения. Каждая переменная предназначена для работы с данными только определенного типа. Соответственно, у каждой переменной есть тип.

Чтобы использовать переменную в программе, ее необходимо объявить. При объявлении переменной указывается тип переменной и ее название.

```
double alpha=30.3, y=3.3;
int x=3;
char ch='A';
```

Переменные в Java

Объявлять переменную можно в любом месте программы, но до того, как переменная впервые используется. Если объявляется несколько переменных одного типа, то их можно перечислить через запятую после идентификатора типа. Одновременно с объявлением переменной ей можно присвоить значение (инициализировать переменную).

Если переменную объявить и инициализировать с ключевым словом **`final`**, то такая переменная становится ***константой***. Изменить значение константы в процессе выполнения программы нельзя.

```
final double g=9.8;
```

Область доступности переменных определяется блоком, в котором переменная объявлена. Блок, в свою очередь, выделяется парой фигурных скобок { и }.

Базовые типы данных в Java

В **Java** существует четыре группы базовых типов:

- для работы с целыми числами (четыре типа),
- для работы с действительными числами в формате с плавающей точкой (два типа),
- для использования символов (один тип)
- логический тип (один).

Базовые типы данных в Java

Тип данных (название)	Количество байт	Описание	Класс оболочка
byte	1	Целые числа от –128 до 127	Byte
short	2	Целые числа от –32 768 до 32 767	Short
int	4	Целые числа от –2 147 483 648 до 2147 483 647	Integer
long	8	Целые числа в диапазоне от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	Long
float	4	Действительные числа $-3{,}4 \times 10^{-38}$ … $3{,}4 \times 10^{38}$	Float
double	8	Действительные числа (двойной точности) $-1{,}7 \times 10^{-308}$ … $1{,}7 \times 10^{308}$	Double
char	2	Символьные значения. Реализуются символы с кодами от 0 до 65 536	Character
boolean	Зависит от виртуальной машины	Логический тип данных. Переменная этого типа может принимать два значения: true и false	Boolean

Базовые типы данных в Java

"Изучаем Java" – строка

'A' или **'Z'** – символы

123 целое число (по умолчанию **int**)

012 целое восьмеричное и означает десятичное число 10

0x12 целое шестнадцатеричное и означает десятичное число 18

123L -?

12.3 вещественное (по умолчанию **double**)

1.2E-3 вещественное, записано в научной (экспоненциальной) нотации и соответствует числу 1.2×10^{-3}

2.5e7 соответствует числу 2.5×10^{7}

12.3f - ?

Приведение типов в Java

Процесс автоматического преобразования типов подчиняется нескольким базовым правилам:

- типы переменных, входящих в выражение, должны быть совместимыми (например, целое число можно преобразовать в формат действительного числа, но не в строку);
- целевой тип (тип, к которому выполняется приведение) должен быть шире исходного типа, т.е. преобразование должно выполняться без потери данных;
- перед выполнением арифметических операций типы **`byte`**, **`short`** и **`char`** расширяются до типа **`int`**;
- если в выражении есть операнды типа **`long`**, то расширение осуществляется до типа **`long`**;
- если в выражении есть операнды типа **`float`**, то расширение осуществляется до типа **`float`**;
- если в выражении есть операнды типа **`double`**, то расширение осуществляется до типа **`double`**.

Приведение типов в Java

Пример:

```
byte x=1, y=2, z;
```

Если мы теперь вычислим сумму **x+y** и захотим присвоить ее в качестве значения переменной **z**, то возникнет ошибка. Хотя все три переменные относятся к типу **byte**, при вычислении выражения **x+y** выполняется автоматическое преобразование к типу **int**. В результате имеет место попытка присвоить значение типа **int** переменной типа **byte**. Поскольку в **Java** преобразования с возможной потерей значений не допускаются, то и программа с таким кодом не скомпилируется.

Исправим:

```
byte x=1,y=2,z;
z=(byte)(x+y);  // использовано явное приведение типа
```

Командой **(byte)(x+y)** вычисляется сумма значений переменных **x** и **y**, а результат преобразуется к типу **byte**.

Явное приведение типа потенциально опасно, поскольку может приводить к потере значения. Такие ситуации должен отслеживать программист – системой они не отслеживаются.

Основные операторы в Java

Операторы **Java** можно разделить на четыре группы:

- арифметические,
- логические,
- побитовые,
- операторы сравнения.

Арифметические операторы

Арифметические: **+, -, *, /, %, ++, --.**

```
int a=7, b=2;
double x=a/b;
```

результат 3.0, а не 3.5, почему?

Сначала вычисляется выражение a/b. Поскольку операнды целочисленные, выполняется целочисленное деление (результат целочисленного деления 7 на 2 равен 3). И только после этого полученное значение преобразуется к формату **double** и присваивается переменной x.

Исправим:

```
double x=(double)a/b;     // теперь результат 3.5
```

Операторы присваивания

Операторы присваивания:
+=, -=, *=, /= и %=

выражение **a+=b** эквивалентно выражению **a=a+b**,

выражение **a*=b** эквивалентно выражению **a=a*b** и т.д.

Команда **x op=y** эквивалентна команде **x=(тип x)(x op y)** — после вычисления выражения (**x op y**) оно еще и приводится к типу переменной **x**, после чего полученное значение присваивается переменной **x**.

Например:

```
char x='A'; // хотим чтобы в x лежал символ 'B'
x=(char)(x+1); //используем оператор явного преобразования типа

или x+=1; // приведение типа происходит автоматически.
```

Инкремент и декремент

Операторы инкремента и декремента **++** и **--**:

Пример:

```
int n,m; n=100; m=n++; // m=?
int n,m; n=100; m=++n; // m=?
```

В первом случае **m=100**, а во-втором **m=101**.

Логические операторы

& бинарный оператор ***логическое и***. Результатом операции вида **A&B** является значение **true**, если значения обоих операндов **A** и **B** равны **true**, в противном случае возвращается значение **false**.

&& бинарный оператор ***логические и*** (сокращенная форма). Особенность оператора, по сравнению с оператором **&**, состоит в том, что если значение первого операнда равно **false**, то значение второго операнда не проверяется.

| бинарный оператор ***логическое или***. Результатом операции вида **A|B** является значение **true**, если значение хотя бы одного из операндов **A** и **B** равно **true**, иначе **false**.

|| бинарный оператор ***логическое или*** (сокращенная форма). Особенность оператора, по сравнению с оператором **|**, состоит в том, что если значение первого операнда равно **true**, то значение второго операнда не проверяется.

Логические операторы

^ бинарный оператор ***исключающее или***.

Результатом операции вида **A^B** является значение **true**, если значение одного и только одного операнда **A** или **B** равно **true**, в противном случае возвращается значение **false**.

! унарный оператор ***логическое отрицание***.

Результатом выражения вида **!A** является значение **true**, если значение операнда **A** равно **false**. Если значение операнда **A** равно **true**, то результатом выражения **!A** является значение **false**.

Операторы сравнения

== оператор равно

Результатом выражения вида **`A==B`** является значение **`true`**, если операнды **`A`** и **`B`** имеют одинаковые значения. В противном случае значением является **`false`**.

< оператор меньше

<= оператор меньше или равно

> оператор больше

>= оператор больше или равно

!= оператор не равно

Побитовые операторы

& бинарный оператор ***побитовое и***. Результатом выражения вида **a&b** с целочисленными операндами **a** и **b** является целое число. Биты в этом числе вычисляются сравнением соответствующих битов в представлении значений операндов **a** и **b**. Если каждый из двух сравниваемых битов равен **1**, то результатом (значение бита) является **1**. В противном случае соответствующий бит равен **0**.

| бинарный оператор ***побитовое или***. Результатом выражения вида **a|b** с целочисленными операндами **a** и **b** является целое число. Биты в этом числе вычисляются сравнением соответствующих битов в представлении значений операндов **a** и **b**. Если хотя бы один из двух сравниваемых битов равен **1**, то результатом (значение бита) является **1**. В противном случае результат (значение бита) равен **0**.

Побитовые операторы

^ бинарный оператор ***побитовое исключающее или***. Результатом выражения вида **a^b** с целочисленными операндами **a** и **b** является целое число. Биты в числе-результате вычисляются сравнением соответствующих битов в представлении значений операндов **a** и **b**. Если один и только один из двух сравниваемых битов равен **1**, то результатом (значением бита) является **1**. В противном случае результат (значение бита) равен **0**.

~ унарный оператор ***побитовая инверсия***. Результатом выражения вида **~a** является целое число, бинарный код которого получается заменой в бинарном коде операнда a нулей на единицы и единиц на нули.

>> бинарный оператор ***сдвиг вправо***. Результатом выражения вида **a>>n** является число, получаемое сдвигом вправо в бинарном представлении первого операнда a на количество битов, определяемых вторым операндом **n**. Исходное значение первого операнда при этом не меняется. Младшие биты теряются, а старшие заполняются значением знакового бита.

Побитовые операторы

<< бинарный оператор ***сдвиг влево***. Результатом выражения вида **a<n** является число, получаемое сдвигом влево в бинарном представлении первого операнда **a** на количество битов, определяемых вторым операндом **n**. Исходное значение первого операнда при этом не меняется. Младшие биты заполняются нулями, а старшие теряются.

>>> бинарный оператор ***беззнаковый сдвиг вправо***. Результатом выражения вида **a>>>n** является число, получаемое сдвигом вправо в позиционном представлении первого операнда **a** на количество битов, определяемых вторым операндом **n**. Исходное значение первого операнда при этом не меняется. Младшие биты теряются, а старшие заполняются нулями.

Тернарный оператор

Синтаксис вызова этого оператора следующий:

```
условие ? оператор1 : оператор2;
```

Первым операндом указывается некоторое условие (выражение, возвращающее логическое значение). Если условие истинно (значение **`true`**), тернарный оператор возвращает значение, указанное после вопросительного знака. Если условие ложно (значение **`false`**), тернарный оператор возвращает значение после двоеточия.

Приоритеты операторов

Несколько замечаний по поводу оператора присваивания **=**.

В **Java** оператор присваивания возвращает значение.

Команда вида **x=y** выполняется следующим образом: вычисляется значение **y** и оно записывается в переменную **x**. Это же значение является значением всего выражения **x=y**. Поэтому допустимой является, например, команда вида

x=y=z .

В этом случае значение переменной **z** присваивается сначала переменной **y**, а затем значение выражения **y=z** (оно же значение переменной **y**) присваивается переменной **x**.

Операторы равных приоритетов (за исключением присваивания) выполняются слева направо. В случаях, когда возникают сомнения в приоритете операторов и последовательности вычисления выражений, рекомендуется использовать круглые скобки.

Приоритеты операторов

Приоритет	Операторы
1	*Круглые скобки* ( ), *квадратные скобки* [ ] и оператор *«точка»*
2	*Инкремент* ++, *декремент* --, *побитовая инверсия* ~ и *логическое отрицание* !
3	*Умножение* *, *деление* / и *вычисление остатка* %
4	*Сложение* + и *вычитание* -
5	Побитовые *сдвиги* >>, << и >>>
6	*Больше* >, *больше или равно* >=, *меньше или равно* <= и *меньше* <
7	*Равно* == и *не равно* !=
8	*Побитовое и* &
9	*Побитовое исключающее или* ^
10	*Побитовое или* |
11	*Логическое и* &&
12	*Логическое или* ||
13	*Тернарный оператор* ?:
14	*Присваивание* = и *составные операторы* вида op=

Использование основных операторов

Составим программу для вычисления координат тела, брошенного под углом к горизонту. Полагаем, что известны масса тела **m**, начальная скорость **V**, угол **α**, под которым тело брошено к горизонту. Кроме того, считаем, что на тело действует сила сопротивления воздуха, по модулю пропорциональная скорости тела и направленная противоположно к направлению полета тела. Коэффициент пропорциональности для силы сопротивления воздуха **γ** также считаем известным. В программе используем известное аналитическое решение для зависимостей координат тела от времени. В частности, для горизонтальной координаты (расстояние от точки бросания до тела вдоль горизонтали) имеем зависимость:

$$x(t) = \frac{Vm\cos(\alpha)}{\gamma}\left(1 - \exp\left(-\frac{\gamma t}{m}\right)\right).$$

Использование основных операторов

Для вертикальной координаты (высота тела над горизонтальной поверхностью) зависимость следующая:

$$y(t) = \frac{m(V\sin(\alpha)\gamma + mg)}{\gamma^2}\left(1 - \exp\left(-\frac{\gamma t}{m}\right)\right) - \frac{mgt}{\gamma}.$$

Здесь через **g** обозначено ускорение свободного падения. Этими соотношениями воспользуемся при создании программы:

Использование основных операторов

```
import static java.lang.Math.*; // Статический импорт
public class HelloWorld {
   public static void main(String[] args) {

   final double g=9.8; // Ускорение свободного падения
   double alpha=30;    // Угол к горизонту (в градусах)
   double m=0.1;       // Масса тела (в килограммах)
   double gamma=0.1;// Коэфф. сопротивл. воздуха (Н*с/м)
   double V=100.0;     // Скорость тела (м/с)
   double t=1.0;       // Время (в секундах)
   double x,y;         // Координаты тела (в метрах)

     alpha/=180/PI;    // Перевод градусов в радианы
    // Вычисление координат:
    x=V*m*cos(alpha)/gamma*(1-exp(-gamma*t/m));
    y=m*(V*sin(alpha)*gamma+m*g)/gamma/gamma*
            (1-exp(-gamma*t/m))-m*g*t/gamma;
    // Вывод информации на экран:
```

Использование основных операторов

```
// Вывод информации на экран:
     System.out.println( "Координаты тела для t="+t+"
сек:\nx="+x+" м\ny="+y+" м");
     System.out.println("Параметры:");
     System.out.println( "Угол alpha="+alpha/PI*180+"
градусов" );
     System.out.println("Скорость V="+V+" м/с");
     System.out.println( "Коэффициент сопротивления
gamma="+gamma+" Н*с/м" );
     System.out.println("Масса тела m="+m+" кг");
   }
 }
```

Результат выполнения:

```
Координаты тела для t=1.0 сек:
x=54.74324621999467 м
y=28.000809417947753 м
Параметры:
Угол alpha=30.0 градусов
Скорость V=100.0 м/с
Коэффициент сопротивления gamma=0.1 Н*с/м
Масса тела m=0.1 кг
```

Использование основных операторов

Класс **Math** доступен в программе и без импорта. Статический импорт используется для того, чтобы не указывать название класса при вызове методов. Другими словами, если не использовать статический импорт, то методы из класса пришлось бы вызывать в формате: **Math.sin**(), **Math.cos**() и **Math.exp**().

Кроме статических методов, в классе **Math** описана константа **PI** со значением числа **π ≈ 3,141592**.

Использование основных операторов

В следующем примере программными методами решается уравнение вида

$$a\cos(x) + b\sin(x) = c.$$

В программе, представленной далее, по значениям параметров a, b и c вычисляется решение (разумеется, если оно существует) уравнения для случая n = 0:

$$x = \arcsin\left(\frac{c}{\sqrt{a^2+b^2}}\right) - \arcsin\left(\frac{a}{\sqrt{a^2+b^2}}\right).$$

При этом проверяется условие существования решения $a^2+b^2 \geq c^2$.

Если данное условие не выполняется, то уравнение решений не имеет.

Введем доп. параметр **`alpha=`** $\cdot \arcsin\left(\frac{a}{\sqrt{a^2+b^2}}\right)$

Использование основных операторов

```
import static java.lang.Math.*;
class HelloWorld{
 public static void main(String args[]){
    double a=5;          // Параметры уравнения
    double b=3;
    double c=1;
    double alpha;        // Вспомогательная переменная
    boolean state;       // критерий наличия решений
 alpha=asin(a/sqrt(a*a+b*b)); // Знач. вспомог. перем.
 state=a*a+b*b>=c*c;  // Вычисление критерия:
 // Вывод на экран значений исходных параметров:
 System.out.println("Уравнение a*cos(x)+b*sin(x)=c");
 System.out.println("Параметры:");
 System.out.println("a="+a);
 System.out.println("b="+b);
 System.out.println("c="+c);
 System.out.print("Решение для x: ");
 // Вычисление решения уравнения и вывод на экран:
 System.out.println( state?asin(c/sqrt(a*a+b*b))-
alpha:"решений нет!" );
  }
 }
```

```
Уравнение a*cos(x)+b*sin(x)=c
Параметры:
a=5.0
b=3.0
c=1.0
Решение для x: -0.8580262366249893
```

Выводы

- **Java** является полностью объектно-ориентированным. Для создания даже самой простой программы необходимо описать по крайней мере один класс. Этот класс содержит главный метод со стандартным названием **main**(). Выполнение программы отождествляется с выполнением главного метода.
- В методе **main**() можно объявлять переменные. Для объявления переменной указывается тип переменной и ее имя. Переменной одновременно с объявлением можно присвоить значение (инициализировать переменную). Переменная должна быть инициализирована до ее первого использования.
- Существует несколько базовых типов данных. При вычислении выражений выполняется автоматическое преобразование типов. Особенность автоматического преобразования типов в **Java** состоит в том, что оно осуществляется без потери значений. Также можно выполнять явное приведение типов, для чего перед выражением в круглых скобках указывается идентификатор соответствующего типа.

Выводы

- Основные операторы **Java** делятся на арифметические, логические, побитовые и операторы сравнения. Арифметические операторы предназначены для выполнения таких операций, как сложение, вычитание, деление и умножение. Логические операторы предназначены для работы с логическими операндами и позволяют выполнять операции отрицания, или, и, исключающего или. Операторы сравнения используются, как правило, при сравнении (на предмет равенства или неравенства) числовых значений. Результатом сравнения является логическое значение (значение логического типа). Побитовые операторы служат для выполнения операций на уровне битовых представлений чисел.
- В **Java** существуют составные операторы присваивания. Команда вида **x = x op y** может быть записана как **x op=y**, где через **op** обозначен арифметический или побитовый оператор.
- В **Java** есть тернарный оператор, который представляет собой упрощенную форму условного оператора. Первым его операндом указывается логическое выражение (условие). В зависимости от значения условия результатом возвращается значение второго или третьего операнда.