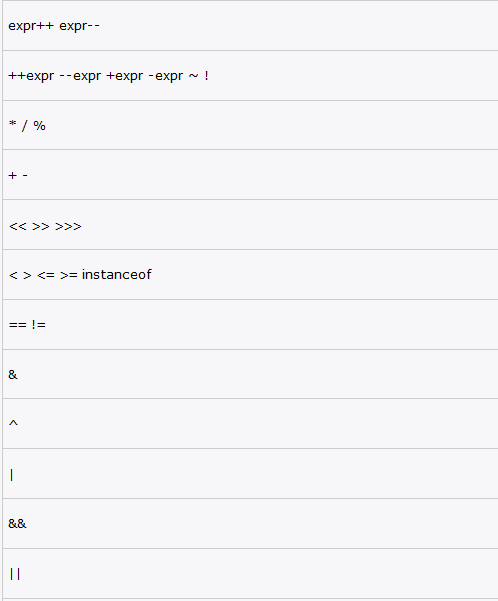
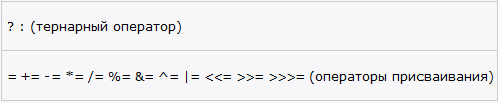
Все операторы Java можно разделить на следующие группы:

* арифметические операторы;
* операторы сравнения;
* побитовые операторы;
* логические операторы;
* операторы присваивания;
* прочие операторы.

|  |  |
| --- | --- |
| Арифметические операторы | |
| + | операция сложения двух чисел |
| - | операция вычитания двух чисел |
| \* | операция умножения двух чисел |
| / | операция деления двух чисел |
| % | получение остатка от деления двух чисел |
| ++ | Префиксный и постфиксный инкремент |
| -- | Префиксный и постфиксный декремент |
| Побитовые операторы | |
| & | побитовое и |
| | | побитовое или |
| ^ | побитовое логическое или |
| ~ | побитовое дополнение |
| << | Побитовый сдвиг влево |
| >> | Побитовый сдвиг вправо |
| >>> | Сдвиг вправо с заполнением нулями |
| операторы сравнения | |
| == | Проверка на равенство |
| != | Проверка на неравенство |
| < | Меньше чем |
| > | Больше чем |
| >= | Больше или равно |
| <= | Меньше или равно |
| логические операторы | |
| | или || | или |
| & или && | и |
| ! | не |
| ^ | Исключающее или |
| Операторы |, & и ||, && идентичны, но во втором случае выражения могут не высчитываться до конца, если их значение уже очевидно | |
| операторы присваивания; | |
| = | обычное |
| += | Сложение с присваиванием |
| -= | Вычитание с присваиванием |
| \*= | Умножение с присваиванием |
| /= | Деление с присваиванием |
| %= | Остаток от деления с присваиванием |
| &= | Логическое и с присваиванием |
| |= | Логическое или с присваиванием |
| ^= | Исключающее или с присваиванием |
| <<= | Сдвиг влево с присваиванием |
| >>= | Сдвиг вправо с присваиванием |
| >>>= | Нулевой сдвиг вправо с присваиванием |

**Приоритет операций**

**УСЛОВНЫЕ КОНСТРУКЦИИ**

If /else

if ([логическое выражение]) {

//do some things

} else {

//do other things

}

  switch/case

switch(var){

case 1:

//do

break;

case 2:

//do

break;

default:

//do

}

Тернарный оператор

int z = x < y ? (x+y) : (x-y);

**ЦИКЛЫ**

for ([инициализация счетчика]; [условие]; [изменение счетчика])

{

    // действия

}

Do while

int j = 7;

do{

    System.out.println(j);

    j--;

}

while (j > 0);

while

int j = 6;

while (j > 0){

    System.out.println(j);

    j--;

}

Оператор **break** позволяет выйти из цикла.

оператор continue позволяет перейти к следующей итерации цикла.

**КЛАССЫ, АБСТРАКТНЫЕ КЛАССЫ, ИНТЕРФЕЙСЫ**

Шаблоном или описанием объекта является **класс**.

**Конструкторы** вызываются при создании нового объекта данного класса и выполняют инициализацию объекта.

Если в классе не определено ни одного конструктора, то для этого класса автоматически создается **конструктор по умолчанию** без параметров.

Оператор **new** выделяет память для объекта.

Ключевое слово **this** представляет ссылку на текущий экземпляр класса.

Кроме конструктора начальную инициализацию объекта вполне можно было проводить с помощью **инициализатора** объекта. Инициализатор выполняется до любого конструктора. То есть в инициализатор мы можем поместить код, общий для всех конструкторов

[модификатор доступа][static] class [имя класса] extends [список наследуемых классов] implements [список реализуемых интерфейсов] {

//поля

//методы и конструктор

//блок инициализации

{

//инициализация

}

}

**Абстрактный класс –** класс, экземпляр которого нельзя создать. призваны предоставлять базовый функционал для классов-наследников. Для создания используется ключевое слово **abstract.**

Производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе. Если класс имеет хотя бы один абстрактный метод, то этот класс должен быть объявлен абстрактным.

**Интерфейсы** определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может применить множество интерфейсов.

Интерфейс может определять константы и методы, которые могут иметь, а могут и не иметь реализации. Если класс реализует интерфейс, то он должен реализовать все методы интерфейса.

Начиная с JDK 8 в интерфейсах доступны статические методы - они аналогичны методам класса.

По умолчанию все методы в интерфейсе фактически имеют модификатор public. Однако начиная с Java 9 мы также можем определять в интерфейсе методы с модификатором private. Они могут быть статическими и нестатическими, но они не могут иметь реализации по умолчанию. Могут использоваться только внутри самого интерфейса, в котором они определены.

Интерфейсы, как и классы, могут наследоваться:

interface BookPrintable extends Printable{

    void paint();

}

Как и классы, интерфейсы могут быть вложенными, то есть могут быть определены в классах или других интерфейсах.

**ВНУТРЕННИЕ И ВЛОЖЕННЫЕ КЛАССЫ**

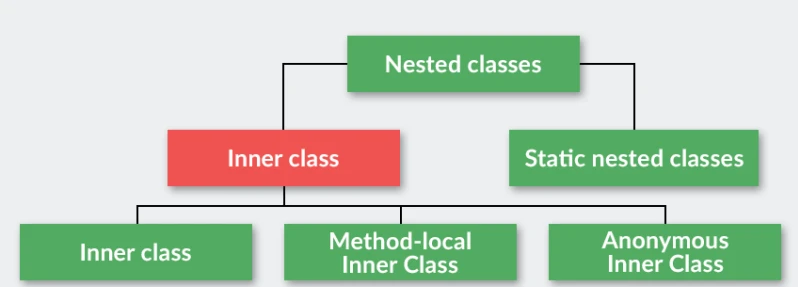
Класс называется **вложенным** (nested), если он определен внутри другого класса. Вложенный класс создается для того, чтобы обслуживать окружающий его класс.

Они делятся на 2 вида:

1. Non-static nested classes — нестатические вложенные классы. По-другому их еще называют inner classes — внутренние классы. Они бывают:

* Внутренний класс (inner class)
* Локальный класс (local class)
* анонимный класс (anonymous class)

1. Static nested classes — статические вложенные классы.



**Особенности внутренних классов**

Объект внутреннего класса не может существовать без объекта «внешнего» класса.

У объекта внутреннего класса есть доступ к переменным «внешнего» класса.

Объект внутреннего класса нельзя создать в статическом методе «внешнего» класса. Во внутренний класс неявно передается ссылка на объект внешнего класса.

Внутренний класс не может содержать статические переменные и методы.

При создании объекта внутреннего класса важную роль играет его модификатор доступа.

Модификаторы доступа для внутренних классов работают так же, как и для обычных переменных.

Можно получить ссылку на экземпляр внешнего класса. Пример: Airplane.this – ссылка на самолет, this – ссылка на крыло.

**Статический вложенный класс**

Для создания экземпляра такого класса, нужно через точку перечислить весь путь от внешнего класса до нужного.

**Локальные классы**

Данные классы объявляются внутри других методов. Они обладают всеми свойствами нестатического вложенного класса, только создавать их экземпляры можно только в методе, при чем метод не может быть статическим.

Локальные классы способны работать только с неизменяемыми переменными (можно не объявлять их final, но они все равно не должны изменяться).

Локальные классы нельзя объявлять с модификаторами доступа.

Локальные классы обладают доступом к переменным метода.

**Анонимные классы –** вложенный класс без имени.

Под капотом анонимные классы – просто обычные нестатические вложенные классы.

Они обычно объявляются либо как подклассы существующего класса, либо как реализации некоторого интерфейса.

Вы можете объявить поля и методы внутри такого класса, но не можете объявить конструктор. Вместо этого возможно объявить статический инициализатор

***Использование анонимных классов оправдано во многих случаях, в частности когда:***

* тело класса является очень коротким;
* нужен только один экземпляр класса;
* класс используется в месте его создания или сразу после него;
* имя класса не важно и не облегчает понимание кода.

**КЛЮЧЕВЫЕ СЛОВА STATIC И FINAL**

В языке программирования Java ключевым словом **static** помечают члены (поля или методы), которые принадлежат классу, а не экземпляру этого класса.

Это означает, что какое бы количество объектов вы не создали, всегда будет создан только один член, доступный для использования всеми экземплярами класса.

Ключевое слово static применимо к переменным, методам, блокам инициализации, импорту и вложенным классам (nested classes).

В языке Java, если поле объявляется статическим, то в независимости от количества созданных объектов класса — всегда будет существовать только один экземпляр статического поля.

Статические переменные размещаются в специальном пуле в памяти JVM, называемом Metaspace. Так как они относятся к классу, статические поля не сериализуются.

Статические переменные могут быть созданы только, как переменные класса. Они не могут быть локальными переменными.

**Статические методы** также принадлежат классу, а не объекту. При этом следует помнить, что из статического метода можно получить доступ только к статическим переменным или к другим статическим методам.

Статические методы можно определять и в интерфейсах, но они не могут быть переопределены методами в реализующих интерфейс классах.

* Абстрактные методы не могут быть статическими
* Статические методы не могут использовать ключевые слова this или super
* Методы экземпляра также могут непосредственно обращаться к статическим переменным и статическим методам
* Статические методы могут обращаться ко всем статическим переменным и другим статическим методам
* Статические поля и методы не являются потокобезопасными
* Статические методы связываются во время компиляции, в отличие от не статических методов, которые связываются во время исполнения. Из-за этого статические методы не могут быть переопределены, т.к. полиморфизм во время выполнения не распространяется на них.

Cтатический импорт позволяет обращаться к статическим членам класса непосредственно по имени члена, без дополнительного указания имени класса и пакета (println).

Статический блок инициализации используется для инициализации статических переменных. Обычно в тех случаях, когда нам требуется выполнить многострочную обработку.

**статические вложенные классы** имеют доступ только к статическим членам внешнего класса.

Суть **модификатора final** - сделать дальнейшее изменение объекта невозможным.

*static final* позволяет объявить константу. Она будет одинаковой в любом экземпляре.

Вы можете применять этот модификатор тремя способами: для класса, для поля (переменной(в том числе аргументам методов)) и для метода.

При использовании с переменной, ее значение нельзя будет изменить после инициализации. Переменные с final - это константы. Для ссылочных переменных это означает, что после присвоения объекта, нельзя изменить ссылку на данный объект. **Это важно!** Ссылку изменить нельзя, но состояние объекта изменять можно.

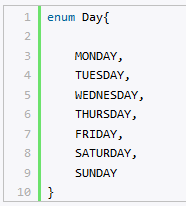
С java 8 появилось понятие — effectively final. Применяется оно только к переменным (в том числе аргументам методов). Суть в том, что не смотря на явное отсутствие ключевого слова final, значение переменной не изменяется после инициализации.

Использование final с методом будет значить, что при наследовании данный метод **нельзя переопределить.**

При использовании final с классом, от данного класса нельзя будет наследоваться.

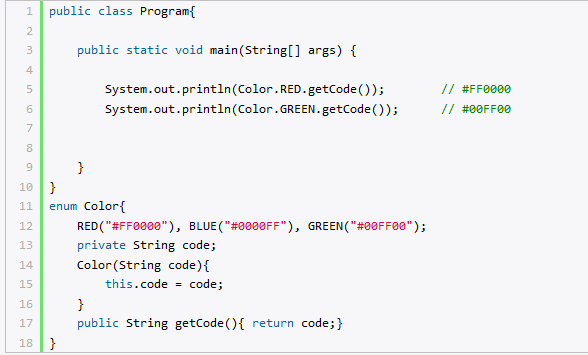
**ENUMS**

Перечисления представляют набор логически связанных констант. Объявление перечисления происходит с помощью оператора enum, после которого идет название перечисления. Затем идет список элементов перечисления через запятую:

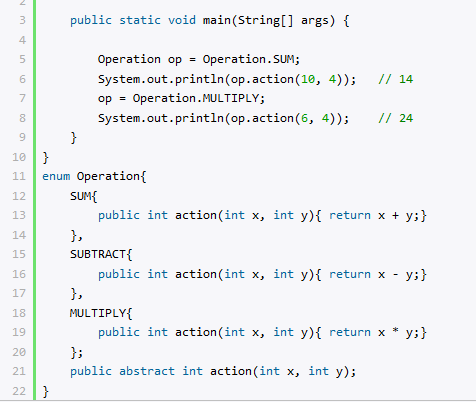


Каждое перечисление имеет статический метод values(). Он возвращает массив всех констант перечисления.

Перечисления, как и обычные классы, могут определять конструкторы, поля и методы. Конструктор всегда приватный, поэтому создавать константы перечисления можно только внутри enum.



Также можно определять методы для отдельных констант:



EnumMap, расширение Enum, Enum в switch.

**ПАКЕТЫ**

Как правило, в Java классы объединяются в пакеты. Пакеты позволяют организовать классы логически в наборы. По умолчанию java уже имеет ряд встроенных пакетов, например, java.lang, java.util, java.io и т.д. Кроме того, пакеты могут иметь вложенные пакеты.

Организация классов в виде пакетов позволяет избежать конфликта имен между классами. Принадлежность к пакету позволяет гарантировать однозначность имен.

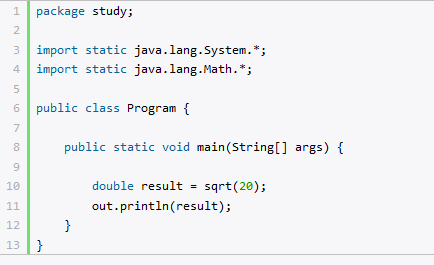
Чтобы указать, что класс принадлежит определенному пакету, надо использовать директиву **package**, после которой указывается имя пакета.

Как правило, названия пакетов соответствуют физической структуре проекта, то есть организации каталогов, в которых находятся файлы с исходным кодом.

Классы необязательно определять в пакеты. Тогда считается что класс находится в пакете по умолчанию.

Если нам надо использовать классы из других пакетов, то нам надо подключить эти пакеты и классы (Можно использовать полный путь с названием всех пакетов). Директива **import** указывается в самом начале кода, после чего идет имя подключаемого класса. Чтобы подключить все классы из пакета можно использовать *packagename.\**.

В java есть также особая форма импорта - статический импорт. Для этого вместе с директивой import используется модификатор static:



Благодаря операции статического импорта мы можем использовать статические методы без названия класса.