# Лекция 7. Модификаторы

# Пространство имён в Java

**Пространство имён** (namespace) — [множество](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE), под которым подразумевается модель, [абстрактное](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D1%8F) [хранилище](https://ru.wikipedia.org/wiki/%D0%A5%D1%80%D0%B0%D0%BD%D0%B8%D0%BB%D0%B8%D1%89%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) или окружение, созданное для [логической](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0) группировки [уникальных](https://ru.wikipedia.org/w/index.php?title=%D0%A3%D0%BD%D0%B8%D0%BA%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C&action=edit&redlink=1) [идентификаторов](https://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%82%D0%BE%D1%80) ([имён](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D1%8F" \o "Имя)). Идентификатор, определенный в пространстве имён, ассоциируется с этим [пространством](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE) и принадлежит ему.

Один и тот же идентификатор может быть независимо определён в нескольких пространствах. Следовательно, иденификаторы, принадлежащие разным пространствам имен, могут имет одинаковое значение.

Языки с поддержкой пространств имён определяют [правила](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C), указывающие, к какому пространству имён принадлежит идентификатор.

В [Java пространства имён представлены пакетами](https://ru.wikipedia.org/wiki/Package_(Java)" \o "Package (Java)) (package). Код всех классов заключен внутри пакетов, при этом код из других пакетов доступен при импортировании (import) необходимо класса (см. формулу 1).

|  |
| --- |
| import полное\_имя\_класса;  //полное\_имя\_классов – то же самое имена\_всех\_его\_пакетов.Имя\_класса |
| Формула 1. Организация импорта классов в Java |

**import** com.tsystems.entity.Dog; //импорт

**public** **class** Main

{

**public** **static** **void** main(String[] args)

{

Dog dog = **new** Dog();

}

}

Пример 1. Использования import для доступа к контексту класса из другого пространства имен

Однако, есть альтернативный вариант импортирования класса – указание полного имени класса в тех местах кода, где данный тип используется.

|  |
| --- |
| //нет импорта  **public** **class** Main  {  **public** **static** **void** main(String[] args)  {  // полное имя класса для импорта  com.tsystems.entity.Dog dog = **new** com.tsystems.entity.Dog();  }  }  Пример 2. Использования полного имени класса для импорта его контекста |
|  |

Пространства имён в Java не являются иерархически упорядоченными из-за синтаксиса самого языка. Тем не менее, пакеты именуются в иерархическом стиле. Например, все пакеты, начинающиеся со слова «java», являются частью [платформы Java](https://ru.wikipedia.org/wiki/Java_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0)).

# Модификаторы в Java

Модификаторы – это ключевые зарезервированные слова языка Java, которые определяют уровень видимости (доступности) класса или его членов (полей, методов). В таблице 1 представлены все модификаторы и их назначение.

|  |  |
| --- | --- |
| Таблица 1. Описание модификаторов в Java | |
| Модификатор | Назначение |
| public | Модификатор дотсупа (всеобщая видимость) |
| protected | Модификатор дотсупа (видимость только наследникам класса) |
| default | Модификатор доступа (виден только внутри пакета, в котором находится) |
| private | Модификатор доступа (виден только в месте объявления) |
| final | Указание неизменности для члена класса |
| abstract | Указание реализации в наслдениках |
| static | Указание контекста класса, а не объекта |

# Модификаторы доступа

В Java существуют следующие модификаторы доступа:

* **public** – всеобщий (доступный из любой области видимости);
* **protected** – защищенный (доступный только наследникам);
* **package-private (default)** – пакетный (доступный только внутри указанного package);
* **private** – закрытый (доступный только внутри объявленной области видимости).

На рисунке 1 приведена область видимости модификаторов доступа – от самого «узкого» до общего.

|  |
| --- |
| 8d3fefbab6d1 |
| Рис. 1. Уровень закрытости модификаторов в Java |

Для членов класса (полей и методов) разрешено использование всех указанных выше модификаторов.

# Модификаторы доступа класса

Класс может быть только либо **public**, либо **default**, причем в одном файле может находиться только один public класс. У нестатического блока может быть только один модификатор – default (при этом, имя самого идентификатора не указывается, так как в Java слово «default» зарезервированно для компонента оператора switch-case).

|  |
| --- |
| **package** com.tsystems.entity; // имя покета, в котором объявлен данный класс  **class** Dog // отсутсвие явного идентификатора указывает компилятора нa package-private  {  } |
| Пример 3. Модификатор default |

В примере 3 представлено использование модификатора default для управления областью видимости класса Dog. В данном случае Dog будет доступен только внутри класса, в котором он объявлен (com.tsystems.entity).

# Модификаторы доступа членов класса

Модификатор доступа у конструкторов, методов и полей может быть любой – public, protected, package-private, private.

Модификаторы доступа для полей:

1. Если поле объявлено как public, значит его измение и доступ к нему доступен в любой точке кода, где создается экземпляр данного класса, и это поле будет доступно для чтения или записи через имя\_переменной.имя\_поля (см. пример 4).

|  |
| --- |
| **package** com.tsystems.entity;  **public** **class** Dog  {  **public** **String** breed; //объявление public-поля  } |
| **import** com.tsystems.entity.Dog; // лежит в другом пакете  **public** **class** Main  {  **public** **static** **void** main(String[] args)  {  Dog myDog=**new** Dog();  System.***out***.println(myDog.breed); //обращение к public-полю  myDog.breed="UNKNOWN"; // изменение поля  System.***out***.println(myDog.breed); }  } |
| Пример 4. public-поле класса Dog |

1. Если поле объявлено, как package-private, то доступ у нему возможет только внутри пакета, в котором лежит класс с этим полем. (**самостоятельно**).
2. Если поле объявлено, как protected, то оно доступно только наслденикам данного класса (**самостоятельно**).
3. Если поле объявлено, как private, то оно не доступно ни вне этого класса, ни в его наслдениках (см. пример 5).

|  |
| --- |
| **package** com.tsystems.entity;  **public** **class** Dog  {  **private** **String** breed; //объявление private-поля  } |
| **package** com.tsystems.entity;  **public** **class** TerrierDog **extends** Dog  {  **protected** **int** age;    **public** TerrierDog()  {  **super**();  // поле breed не доступно наследнику Dog  }  } |
| Пример 4. public-поле класса Dog |

Модификаторы доступа для методов:

1. Если метод имеет модификатор public, то он может быть вызван в любой точке кода на переменной типа класса, которому принадлежит данный метод.
2. Если метод имеет модификатор доступа protected, то его вызов возможен только в наслдениках класса, в котором объявлен данный метод.
3. Если метод имеет модификатор доступа package-private, то он может быть вызван на объекте данного типа класса только внутри пакета, а котором находится этот класс.
4. Если метод имеет метод имеет модификатор доступа private, он может быть вызван только внутри класса, в котором он объявлен.

**Во время наследования возможно изменение модификаторов доступа в сторону БОЛЬШЕЙ видимости.**

# Модификаторы static, abstract и final

# Модификатор static

static-модификатор может применяться к методам, переменным и блокам в классе.

Метод, объявленный как static не принадлежит экземпляру класса, а принадлежит самому классу, и может быть вызыван без создания экземпляра данного класса. Объявление такого метода осуществляется согласно формуле 2, вызов – согласно формуле 3. Статические методы имееют доступ только к статическим полям своего класса.

|  |
| --- |
| [модификатор\_доступа] **static** тип\_возвращаемого\_значения имя\_метода([список\_входных\_парметров]) {} |
| Формула 2. Объявление статического метода |

|  |
| --- |
| Имя\_класса.имя\_статического\_метода([список\_параметров]); |
| Формула 3. Вызов статического метода класса |

В классе могут быть объявлены статические переменные. Они инициализируются во время загрузки класса и, аналогично методам, не принадлежат конкретному экземпляру класса, а принадлежат самому классу. При этом, статические переменные едины для всех объектов данного класса, так как на статическую переменную создается в памяти одна единственная ссылка, доступная всем экземплярам. Объявление и обращение к статической переменной класса осуществляется согласно формуле 4 соотвественно.

|  |
| --- |
| [модификатор\_доступа] **static** тип имя\_статического\_поля; |
| Формула 4. Объявление к статическому полю класса |

|  |
| --- |
| Имя\_класса.имя\_статического\_поля; |
| Формула 5. Обращение к статическому полю класса |

В классе также могут быть объявлены статические блоки инциализации. Они абсолютно идентичны простым блокам инициализации, с той лишь разницей, что их вызов происходит не при инициализации нового экземпляра класса, а на этапе загрузки класса (т.е одиножды).

|  |
| --- |
| **static** {/\*тело статического блока инициализации\*/} |
| Формула 6. Создание статичсекого блока инициализации |

Локальные переменные (переменные внутри методов) не могут быть объявлены как static.

# Модификатор final

final-модификатор служит для явного указания неизменности того, что указано, как final.

1. Если класс декларирован, как final, то он не может быть унаследован и является последним в своей иерархии.
2. Поле класса, указанное как final, должно быть обязательно проинициализировано на этапе создания объекта данного класса любым способом инииализации, и не может быть измненено в процессе жизни объекта.
3. final-метод не может быть изменен в потомках своего класса.
4. Локальная переменная не может быть изменена после задания ей значения.
5. Параметры методов, указанные как final, предназначены для чтения в контесте, в который они передаются, и не могут быть изменены в нем.
6. final-переменные не возможно проинициализировать «по-умолчанию», их значение необходимо задавать в явном виде.
7. Если final переменная содержит ссылку на объект, объект может быть изменен (неизменной является ссылка на этот объект, которую хранит переменная).

# Модификатор abstract

Данный модификатор применяет только для методов и классов, причем, абстрактный метод может быть объявлен только в абстрактоном классе.

Идея создания абстрактоного класса заключается в том, что на высоких уровнях иерархии классов еще не ясно до конца какие именно объекты должны взаимодействовать в программе, однако некие общие черты их выявить уже удалось. Т.е. абстрактные классы – это некоторые заготовки для будущих классов, именно поэтому на основе асбтрактного класса невозможно создать объект. Однако, абстрактный класс может содержать поля и методы. Абстрактные классы созданы только для того, чтобы быть унаследованными простыми классами и «допроектированы» ими до конца.

|  |
| --- |
| **[модификатор\_доступа] abstract class Имя\_абстрактного\_класса**  {/\*тело абстрактного класса\*/} |
| Формула 7. Объявление абстрактного класса |

Зачастую абстрактные классы содержат абстрактные методы – методы без реализации. Такие методы только объявлены в абстрактном классе, а реализованны будут в потомках. Как видно из формулы 6, у абстрактных методов нет тела метода.

|  |
| --- |
| **[модификатор\_доступа] abstract тип\_возвращаемого\_значения имя\_абстрактного\_метода([список\_параметров]);** |
| Формула 7. Объявление абстрактного метода |

Класс должен быть объявлен как abstract, если он:

* содержит хотя бы один абстрактный метод;
* не предоставляет реализацию наследуемых абстрактных методов;
* он не предоставляет реализацию методов интерфейса, реализацию которого он объявил;
* необходимо запретить создание экземпляров класса.

**Конструктор класса** не может быть static, abstract или final !!!

Самостоятельно:

1. Создать иерархию вида (без детального описания полей класса).
2. В классе Entity создать 3 метода – начатьЖить, закончитьЖить, приниматьПищу, при этом метод принимать пищу должен быть абстрактным.
3. Создайте в Entity final-поле isAnimal, отвечающее за то, является ли данная сущность животным.
4. Все наследники Entity должны знать, существуют ли они в мире.
5. Только собаки должны иметь имя.
6. Только животные могут издавать звуки.
7. Класс Entity не должен быть виден нигде, кроме своего пакета.
8. Методы начатьЖить и закончитьЖить класса Entity не должны быть доступны никому, кроме наследников.