Оглавление

[Виды классов 2](#_Toc47521009)

[Вложенный класс (nested class) 2](#_Toc47521010)

[Локальный класс (local inner class) 2](#_Toc47521011)

[анонимный класс (anonymous inner class). 4](#_Toc47521012)

[Каким образом из вложенного класса получить доступ к полю внешнего класса? 6](#_Toc47521013)

[!!! Перечисления (enum) 7](#_Toc47521014)

[Проблема ромбовидного наследования классов и интерфейсов 7](#_Toc47521015)

[Конструкторы по умолчанию, копирования и с параметрами 7](#_Toc47521016)

[Где и как вы можете использовать приватный конструктор? 8](#_Toc47521017)

[Классы-загрузчики и динамическая загрузка классов 8](#_Toc47521018)

[Модификаторы доступа 12](#_Toc47521019)

[static 12](#_Toc47521020)

[Может ли статический метод быть переопределён или перегружен? 13](#_Toc47521021)

[Могут ли нестатические методы перегрузить статические? 13](#_Toc47521022)

[Можно ли сузить уровень доступа/тип возвращаемого значения при переопределении метода? 14](#_Toc47521023)

[Возможно ли при переопределении метода изменить: модификатор доступа, возвращаемый тип, тип аргумента или их количество, имена аргументов или их порядок; убирать, добавлять, изменять порядок следования элементов секции throws? 14](#_Toc47521024)

[Могут ли классы быть статическими? 15](#_Toc47521025)

[final 15](#_Toc47521026)

[Абстрактный класс: 15](#_Toc47521027)

[Может ли быть абстрактный класс без абстрактных методов 16](#_Toc47521028)

[Могут ли быть конструкторы у абстрактных классов? Для чего они нужны? 16](#_Toc47521029)

[Интерфейс (interface) 16](#_Toc47521030)

[Метод интерфейса с модификатором final? 17](#_Toc47521031)

[Абстрактный класс (АК) и интерфейс (И) 17](#_Toc47521032)

[Может ли один интерфейс наследоваться от другого? От двух других? 18](#_Toc47521033)

[Что такое дефолтные методы интерфейсов? Для чего они нужны? 18](#_Toc47521034)

[Как решается проблема ромбовидного наследования при наследовании интерфейсов при наличии default методов? 18](#_Toc47521035)

# Виды классов

∙ Обычный класс (top level class)  
∙ абстрактный класс;  
∙ финализированный класс (final class);

∙ Интерфейс (interface)

∙ Перечисление (enum);

∙ Вложенный класс (nested class):  
∙ статический класс (static nested class);  
∙ внутренний класс (inner class): простой внутренний класс (member inner class), локальный класс (local inner class), анонимный класс (anonymous inner class).

# Вложенный класс (nested class)

∙ определен внутри другого класса;  
∙ должен обслуживать только внешний класс;  
∙ имеет доступ к полям и методам внешнего класса.

# Локальный класс (local inner class)

Вложенный класс, который может быть декларирован в любом блоке, в котором разрешается декларировать переменные. Как и простые внутренние классы локальные классы имеют имена и могут использоваться многократно. Как и анонимные классы, они имеют окружающий их экземпляр только тогда, когда применяются в нестатическом контексте.

Особенности:  
∙ видны только в пределах блока, в котором объявлены;  
∙ не могут быть объявлены как private/public/protected или static;  
∙ Не могут иметь внутри себя статических объявлений методов и классов, но могут иметь финальные статические поля, проинициализированные константой;  
∙ имеют доступ к полям и методам обрамляющего класса;  
∙ могут обращаться к локальным переменным и параметрам метода, если они объявлены с модификатором final.

объявляется внутри блока кода.

Особенности:

не имеет модификаторов доступа;

не может иметь внутри себя статических объявлений методов и классов

может иметь final static поля;

может обращаться к локальным переменным и параметрам метода, если они final.

еще!!!

# анонимный класс (anonymous inner class).

Это вложенный локальный класс без имени, который разрешено декларировать в любом месте обрамляющего класса, разрешающем размещение выражений. Создание экземпляра анонимного класса происходит одновременно с его объявлением. В зависимости от местоположения анонимный класс ведет себя как статический либо как нестатический вложенный класс — в нестатическом контексте появляется окружающий его экземпляр.

Анонимные классы имеют несколько ограничений:  
∙ их использование разрешено только в одном месте программы — месте его создания;  
∙ применение возможно только в том случае, если после порождения экземпляра нет необходимости на него ссылаться;  
∙ реализует лишь методы своего интерфейса или суперкласса, т. е. не может объявлять каких-либо новых методов, т. к. для доступа к ним нет поименованного типа.

Анонимные классы обычно применяются для:  
∙ создания объекта функции (function object), например, реализация интерфейса Comparator;  
∙ создания объекта процесса (process object), такого как экземпляры классов Thread, Runnable и подобных;  
∙ в статическом методе генерации;  
∙ инициализации открытого статического поля final, которое соответствует сложному перечислению типов, когда для каждого экземпляра в перечислении требуется отдельный подкласс.

без имени;

разное поведение в зависимости от местоположения (статический или внутренний классы)

Ограничения:

Их использование разрешено только в одном месте программы — месте его создания;

Применение возможно только в том случае, если после порождения экземпляра нет необходимости на него ссылаться;

Реализует лишь методы своего интерфейса или суперкласса, т. е. не может объявлять каких либо новых методов, так как для доступа к ним нет поименованного типа.

Применяются для:

создания объекта функции (function object), например реализация интерфейса Comparator;

создания объекта процесса (process object), такого как экземпляры классов Thread, Runnable и подобных;

в статическом методе генерации;

инициализации открытого статического поля final, которое соответствует сложному перечислению типов, когда для каждого экземпляра в перечислении требуется отдельный подкласс.

## Каким образом из вложенного класса получить доступ к полю внешнего класса?

Статический вложенный класс имеет прямой доступ только к статическим полям обрамляющего класса.

Простой внутренний класс, может обратиться к любому полю внешнего класса напрямую. В случае, если у вложенного класса уже существует поле с таким же литералом, то обращаться к такому полю следует через ссылку на его экземляр. Например: Outer.this.field.

## !!! Перечисления (enum)

# Проблема ромбовидного наследования классов и интерфейсов

Запрещено множественное наследование у классов, переопределение дефолтного метода интерфейса.

# Конструкторы по умолчанию, копирования и с параметрами

Конструктор — специальный метод, у которого отсутствует возвращаемый тип и который имеет то же имя, что и класс, в котором он используется. Конструктор вызывается при создании нового объекта класса и определяет действия необходимые для его инициализации.

Если у какого-либо класса не определить конструктор, то компилятор сгенерирует конструктор без аргументов — конструктор по умолчанию.

Если у класса уже определен какой-либо конструктор, то конструктор по умолчанию создан не будет и, если он необходим, его нужно описывать явно.

У конструктора по умолчанию отсутствуют какие-либо аргументы. Конструктор копирования принимает в качестве аргумента уже существующий объект класса для последующего создания его клона. Конструктор с параметрами имеет в своей сигнатуре аргументы (обычно необходимые для инициализации полей класса).

# Где и как вы можете использовать приватный конструктор?

∙ вложенными классами.

Приватный конструктор может использоваться публичным статическим методом генерации объектов данного класса.

# Классы-загрузчики и динамическая загрузка классов

Основа работы с классами в Java — классы-загрузчики, обычные Java-объекты, предоставляющие интерфейс для поиска и создания объекта класса по его имени во время работы приложения.

В начале работы программы создается 3 основных загрузчика классов:

∙ Базовый загрузчик. Загружает основные системные и внутренние классы JDK (Core API — пакеты java.\* (rt.jar и i18n.jar)). Важно заметить, что базовый загрузчик является «Изначальным» или «Корневым» и частью JVM, вследствие чего его нельзя создать внутри кода программы.

∙ Загрузчик расширений (extention). Загружает различные пакеты расширений, которые располагаются в директории <JAVA\_HOME>/lib/ext или другой директории, описанной в системном параметре java.ext.dirs. Это позволяет обновлять и добавлять новые расширения без необходимости модифицировать настройки используемых приложений. Загрузчик расширений реализован классом sun.misc.Launcher$ExtClassLoader.

∙ Системный загрузчик (system/application). Загружает классы, пути к которым указаны в переменной окружения CLASSPATH или пути, которые указаны в командной строке запуска JVM после ключей -classpath или -cp. Системный загрузчик реализован классом sun.misc.Launcher$AppClassLoader.

Загрузчики классов являются иерархическими: каждый из них (кроме базового) имеет родительский загрузчик и в большинстве случаев, перед тем как попробовать загрузить класс самостоятельно, он посылает вначале запрос родительскому загрузчику загрузить указанный класс. Такое делегирование позволяет загружать классы тем загрузчиком, который находится ближе всего к базовому в иерархии делегирования. Как следствие поиск классов будет происходить в источниках в порядке их доверия: сначала в библиотеке Core API, потом в папке расширений, потом в локальных файлах CLASSPATH.

Процесс загрузки класса состоит из трех частей:

∙ Loading — на этой фазе происходит поиск и физическая загрузка файла класса в определенном источнике (в зависимости от загрузчика). Этот процесс определяет базовое представление класса в памяти. На этом этапе такие понятия как «методы», «поля» и т. д. пока не известны.

∙ Linking — процесс, который может быть разбит на 3 части:  
∙ Bytecode verification — проверка байт-кода на соответствие требованиям, определенным в спецификации JVM.  
∙ Class preparation — создание и инициализация необходимых структур, используемых для представления полей, методов, реализованных интерфейсов и т. п., определенных в загружаемом классе.  
∙ Resolving — загрузка набора классов, на которые ссылается загружаемый класс.

∙ Initialization — вызов статических блоков инициализации и присваивание полям класса значений по умолчанию.

Динамическая загрузка классов в Java имеет ряд особенностей:  
∙ отложенная (lazy) загрузка и связывание классов. Загрузка классов производится только при необходимости, что позволяет экономить ресурсы и  распределять нагрузку.  
∙ проверка корректности загружаемого кода (type safeness). Все действия связанные с контролем использования типов производятся только во время загрузки класса, позволяя избежать дополнительной нагрузки во время выполнения кода.  
∙ программируемая загрузка. Пользовательский загрузчик полностью контролирует процесс получения запрошенного класса — самому ли искать байт-код и создавать класс или делегировать создание другому загрузчику. Дополнительно существует возможность выставлять различные атрибуты безопасности для загружаемых классов, позволяя таким образом работать с кодом из ненадежных источников.  
∙ множественные пространства имен. Каждый загрузчик имеет свое пространство имен для создаваемых классов. Соответственно, классы, загруженные двумя различными загрузчиками на основе общего байт-кода, в системе будут различаться.

Существует несколько способов инициировать загрузку требуемого класса:  
∙ явный: вызов ClassLoader.loadClass() или Class.forName() (по умолчанию используется загрузчик, создавший текущий класс, но есть возможность и явного указания загрузчика);  
∙ неявный: когда для дальнейшей работы приложения требуется ранее не использованный класс, JVM инициирует его загрузку.

# Модификаторы доступа

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| название | видимость членов классов | | | | видимость класса | модифи-катор |
| внутри класса | внутри пакета | наслед-  никам | всем |
| приватный | да | нет | нет | нет | — | private |
| доступ на уровне пакета | да | да | нет | нет | + |  |
| защищенный | да | да | да | нет | — | protected |
| публичный | да | да | да | да | + | public |

При наследовании можно изменять модификаторы доступа в сторону большей видимости (принцип подстановки Барбары Лисков).

# static

∙ полям;  
∙ методам;  
∙ вложенным классам;  
∙ блокам инициализации;  
∙ членам секции import.

# Может ли статический метод быть переопределен или перегружен?

Перегружен — да. Все работает точно так же, как и с обычными методами — два статических метода могут иметь одинаковое имя, если количество их параметров или типов различается.

Переопределен — нет. Выбор вызываемого статического метода происходит при раннем связывании (на этапе компиляции, а не выполнения) и выполняться всегда будет родительский метод, хотя синтаксически переопределение статического метода — это вполне корректная языковая конструкция.

В целом, к статическим полям и методам рекомендуется обращаться через имя класса, а не объект.

# Могут ли нестатические методы перегрузить статические?

Да. В итоге получится два разных метода. Статический будет принадлежать классу и будет доступен через его имя, а нестатический будет принадлежать конкретному объекту и доступен через вызов метода этого объекта.

# Можно ли сузить уровень доступа/тип возвращаемого значения при переопределении метода?

# Возможно ли при переопределении метода изменить: модификатор доступа, возвращаемый тип, тип аргумента или их количество, имена аргументов или их порядок; убирать, добавлять, изменять порядок следования элементов секции throws?

При переопределении метода сужать модификатор доступа не разрешается, т. к. это приведёт к нарушению принципа подстановки Барбары Лисков. Расширение уровня доступа возможно.

Можно изменять все, что не мешает компилятору понять какой метод родительского класса имеется в виду:

Изменять тип возвращаемого значения при переопределении метода разрешено только в сторону сужения типа (вместо родительского класса — наследника).

При изменении типа, количества, порядка следования аргументов вместо переопределения будет происходить overloading (перегрузка) метода.

Секцию throws метода можно не указывать, но стоит помнить, что она остается действительной, если уже определена у метода родительского класса. Так же, возможно добавлять новые исключения, являющиеся наследниками от уже объявленных или исключения RuntimeException. Порядок следования таких элементов при переопределении значения не имеет.

# Могут ли классы быть статическими?

Только вложенные.

# final

∙ класс не может иметь наследников;  
∙ метод не может быть переопределен;  
∙ поле и локальные переменные не могут быть изменены после инициализации;  
∙ параметры методов не могут изменять свое значение.

# Абстрактный класс:

∙ не может иметь экземпляры;  
∙ выступает только предком;  
∙ наследниками могут быть любые (абстрактные и не абстрактные классы).

Абстрактный метод — метод без реализации.

Если в классе присутствует хотя бы один абстрактный метод, то весь класс должен быть объявлен абстрактным.

Абстрактные классы и методы описывают шаблон объекта, который должен быть реализован в других классах.

# Может ли быть абстрактный класс без абстрактных методов

Да.

# Могут ли быть конструкторы у абстрактных классов? Для чего они нужны?

Да, для наследования.

# Интерфейс (interface)

Контракт класса (правила взаимодейтсвия), описывающий возможности.

Методы:  
∙ без модификатора — неявно public и abstract;  
∙ default — обычные методы (Java 8);  
∙ static (Java 8).

Поля — неявно public, static и final (Java 8).

# Метод интерфейса с модификатором final?

Нет. Он неявно абстрактный и дожен быть переопределен, а final это делать запрещает.

# Абстрактный класс (АК) и интерфейс (И)

Их количество и ключевые слова при наследовании/реализации:  
∙ наследоваться от одного АК (extends);  
∙ реализовать несколько И (implements).

Наличие связи с классом:  
∙ АК — отношений is a (является);  
∙ И — без связи.

Наличие конструктора:  
∙ АК — да (для наследников);  
∙ И — нет.

Назначение:  
АК — инструмент частичной реализации поведения (позволяет сократить код);  
∙ И — контракт класса (правила взаимодействия), описывающим возможности (набор доступных методов).

Методы:  
∙ АК: abstract, обычные и static.  
∙ И: без модификатора (неявно public и abstract), default (обычные методы классов (Java 8)) и static (Java 8).

АК «растворяет» индивидуальность, И расширяет функциональность.

И позволяет создавать структуры типов без иерархии.

Т. о., АК применяются в случае построения иерархии однотипных, очень похожих друг на друга классов. В остальных случаях лучше использовать интерфейсы.

Пример: животные (АК) и летать (птицы, самолет) (И).

# Может ли один интерфейс наследоваться от другого? От двух других?

Да, множественное наследование интерфейсов не запрещено.

# Что такое дефолтные методы интерфейсов? Для чего они нужны?

Аналогия обычных методов классов для интерфейсов.

# Как решается проблема ромбовидного наследования при наследовании интерфейсов при наличии default методов?

Метод должен быть обязательно переопределен.

# Каков порядок вызова конструкторов и блоков инициализации с учетом иерархии классов?

Parent static block(s) → Child static block(s) → Grandchild static block(s)

Parent non-static block(s) → Parent constructor → Child non-static block(s) → Child constructor → Grandchild non-static block(s) → Grandchild constructor

# Зачем нужны и какие бывают блоки инициализации?

Блоки инициализации представляют собой код, заключенный в фигурные скобки и размещаемый внутри класса вне объявления методов или конструкторов.

∙ Существуют статические и нестатические блоки инициализации.

∙ Блок инициализации выполняется перед инициализацией класса загрузчиком классов или созданием объекта класса с помощью конструктора.

∙ Несколько блоков инициализации выполняются в порядке следования в коде класса.

∙ Блок инициализации способен генерировать исключения, если их объявления перечислены в throws всех конструкторов класса.

∙ Блок инициализации возможно создать и в анонимном классе.

# Для чего в Java используются статические блоки инициализации?

Статические блоки инициализация используются для выполнения кода, который должен выполняться один раз при инициализации класса загрузчиком классов, в момент, предшествующий созданию объектов этого класса при помощи конструктора. Такой блок (в отличие от нестатических, принадлежащих конкретном объекту класса) принадлежит только самому классу (объекту метакласса Class).

# Что произойдет, если в блоке инициализации возникнет исключительная ситуация?

Для нестатических блоков инициализации, если выбрасывание исключения прописано явным образом требуется, чтобы объявления этих исключений были перечислены в throws всех конструкторов класса. Иначе будет ошибка компиляции. Для статического блока выбрасывание исключения в явном виде, приводит к ошибке компиляции.

В остальных случаях, взаимодействие с исключениями будет проходить так же как и в любом другом месте. Класс не будет инициализирован, если ошибка происходит в статическом блоке и объект класса не будет создан, если ошибка возникает в нестатическом блоке.

маркерный интерфейс</div>

отсутствуют методы

относит класс к определенному типу (Clonable).

Статический класс (static nested class);

не нужна связь между объектами вложенного и внешнего классов.

static;

только вложенный класс.

Внутренний класс

нужна связь между объектами вложенного и внешнего классов.

более высокий уровень абстракции — класс, абстрактный класс или интерфейс?

Интерфейс.

статический и внутренний классы

Для создания объекта статического вложенного класса объект внешнего класса не требуется.

Из объекта статического вложенного класса нельзя обращаться к не статическим членам обрамляющего класса напрямую, а только через ссылку на экземпляр внешнего класса.

Обычные вложенные классы не могут содержать статических методов, блоков инициализации и классов. Статические вложенные классы — могут.

В объекте обычного вложенного класса хранится ссылка на объект внешнего класса. Внутри статического такой ссылки нет. Доступ к экземпляру обрамляющего класса осуществляется через указание .this после его имени. Например: Outer.this.

внутренний класс (inner class).

доступ к членам класса

доступ к приватному члену класса

Доступ к приватной переменной

открыт внутри класса (вложенные классы);

извне — геттеры и сеттеры (не приватные методы) и механизм рефлексии (Reflection API).

Каким образом из вложенного класса получить доступ к полю внешнего класса?

Статический вложенный класс имеет прямой доступ только к статическим полям обрамляющего класса.

Простой внутренний класс, может обратиться к любому полю внешнего класса напрямую. В случае, если у вложенного класса уже существует поле с таким же литералом, то обращаться к такому полю следует через ссылку на его экземляр. Например: Outer.this.field.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6. | Stream fromBuilder = Stream.builder()  .add("x")  .add("y")  .add("z")  .build();  fromBuilder.forEach(System.out::println); |

asdfasdf