Оглавление

[1. Дайте определение таким понятиям как” класс” и “объект”. Приведите примеры объявления класса и создания объекта класса. Какие спецификаторы можно использовать при объявлении класса. 2](#_Toc20830867)

[2. Как вы определяете, какие поля и методы необходимо определить в классе, приведите пример. Какие спецификаторы можно использовать с полями, а какие с методами (и что они значат)? 3](#_Toc20830868)

[3. Что такое конструктор? Как вы отличите конструктор от любого другого метода? Сколько конструкторов может быть в классе? Что такое конструктор по умолчанию, может ли в классе совсем не быть конструкторов? Объясните, что делает оператор this() в конструкторе? 6](#_Toc20830869)

[4. Опишите процедуру инициализации полей класса и полей экземпляра класса. Когда инициализируются поля класса, а когда – поля экземпляров класса. Какие значения присваиваются полям по умолчанию? Где еще в классе полям могут быть присвоены начальные значения? 6](#_Toc20830870)

[5. Дайте определение перегрузке методов. Как вы думаете, чем удобна перегрузка методов? Укажите, какие методы могут перегружаться, и какими методами они могут быть перегружены? Можно ли перегрузить методы в базовом и производном классах? Можно ли private метод базового класса перегрузить public методов производного? Можно ли перегрузить конструкторы, и можно ли при перегрузке конструкторов менять атрибуты доступа у конструкторов? 6](#_Toc20830871)

[6. Объясните, что такое раннее и позднее связывание? Перегрузка – это раннее или позднее связывание? Объясните правила, которым следует компилятор при разрешении перегрузки; в том числе, если методы перегружаются примитивными типами, между которыми возможно неявное приведение или ссылочными типами, состоящими в иерархической связи. 6](#_Toc20830872)

[7. Объясните, как вы понимаете, что такое неявная ссылка this? В каких методах эта ссылка присутствует, а в каких – нет, и почему? 6](#_Toc20830873)

[8. Что такое финальные поля, какие поля можно объявить со спецификатором final? Где можно инициализировать финальные поля? 6](#_Toc20830874)

[9. Что такое статические поля, статические финальные поля и статические методы. К чему имеют доступ статические методы? Можно ли перегрузить и переопределить статические методы? Наследуются ли статические методы? 7](#_Toc20830875)

[10. Что такое логические и статические блоки инициализации? Сколько их может быть в классе, в каком порядке они могут быть размещены и в каком порядке вызываются? 7](#_Toc20830876)

[11. Что представляют собой методы с переменным числом параметров, как передаются параметры в такие методы и что представляет собой такой параметр в методе? Как осуществляется выбор подходящего метода, при использовании перегрузки для методов с переменным числом параметров? 7](#_Toc20830877)

# Дайте определение таким понятиям как” класс” и “объект”. Приведите примеры объявления класса и создания объекта класса. Какие спецификаторы можно использовать при объявлении класса.

Класс - содержание и поведение некой совокупности данных и действий над этими данными. Объявление класса производится с помощью ключевого слова class. Классы могут наследовать методы и данные один другого, кроме конструкторов и инициализаторов. Наследование реализуется с помощью ключевого слова extends (class <имя\_класса> extends <имя\_суперкласса>). В то же время при желании любой из наследников может перегрузить унаследованное свойство.

Правила работы с классами:

* В исходном файле может быть только один публичный класс (public class).
* Исходный файл может иметь несколько "непубличных" классов.
* Название публичного класса должно совпадать с именем исходного файла, который должен иметь расширение .java в конце. Например: имя класса public class Employee{}, то исходный файл должен быть Employee.java.
* Если класс определен внутри пакета, то оператор пакет должно быть первым оператором в исходном файле.
* Если присутствуют операторы импорта, то они должны быть написаны между операторами пакета и объявлением класса. Если нет никаких операторов пакета, то оператор импорта должен быть первой строкой в исходном файле.
* Операторы импорта и пакета будут одинаково выполняться для всех классов, присутствующих в исходном файле. В Java не представляется возможным объявить различные операторы импорта и/или пакета к различным классам в исходном файле.

Class Mycl{ // пример объявления класса

// методы

// данные

}

Модификаторы:

Модификатор **abstract**, примененный к классу, говорит о том, что класс является (или считается) незаконченным, а задание "завершить" класс возлагается на наследников. Попытка инстанциировать такой класс приведет к ошибке компиляции.

Модификатор **final** у класса говорит о том, что от него нельзя наследоваться.  Класс нельзя объявить одновременно final и abstract.

Вложенные классы в Java могут быть объявлены как **static.** В этом случае класс называется статическим вложенным классом и имеет доступ к статическим полям и методам обрамляющего класса.

Объект-  некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением, имеющая определенные свойства ([атрибуты](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0)) и операции над ними ([методы](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))). Для создание объекта, то необходимо создать класс, содержащей метод main(), внутри которого и происходит выполнение программы, в котором создать объект с помощью оператора new.

Создание объекта состоит из трёх этапов:

* Декларация  
  Декларирование переменной типа класса с уникальным именем.
* Инстанциация  
  Создание нового объекта с использование ключевого слова ‘new’.
* Инициализация  
  Вызов конструкторв, который идёт после ключевого слова ‘new’.

# Как вы определяете, какие поля и методы необходимо определить в классе, приведите пример. Какие спецификаторы можно использовать с полями, а какие с методами (и что они значат)?

В программе необходимо определять те методы и поля, что отвечает логики программы, предметной области, логики и ООП. Например, в классе ноутбук логично использовать методы включить и выключить. В качестве полей модель и производитель.

Модификаторы:

#### **Модификатор static.**

#### Переменные static

#### Ключевое слово static используется для создания переменных, которые будут существовать независимо от каких-либо экземпляров, созданных для класса. Только одна копия переменной static в Java существует вне зависимости от количества экземпляров класса.

#### Статические переменные также известны как переменные класса. В Java локальные переменные не могут быть объявлены статическими (static).

#### Методы static Ключевое слово static используется для создания методов, которые будут существовать независимо от каких-либо экземпляров, созданных для класса. В Java статические методы или методы static не используют какие-либо переменные экземпляра любого объекта класса, они определены. Методы static принимают все данные из параметров и что-то из этих параметров вычисляется без ссылки на переменные. Переменные и методы класса могут быть доступны с использованием имени класса, за которым следует точка и имя переменной или метода.

* **Модификатор final** — используется для завершения реализации классов, методов и переменных.

Переменная final может быть инициализирована только один раз. Ссылочная переменная, объявленная как final, никогда не может быть назначен для обозначения другого объекта. Однако данные внутри объекта могут быть изменены. Таким образом, состояние объекта может быть изменено, но не ссылки. С переменными в Java модификатор final часто используется со static, чтобы сделать константой переменную класса.

Метод final не может быть переопределен любым подклассом. Как упоминалось ранее, в Java модификатор final предотвращает метод от изменений в подклассе. Главным намерение сделать метод final будет то, что содержание метода не должно быть изменено стороне.

Класс final. Основная цель в Java использования класса объявленного в качестве final заключается в предотвращении класс от быть подклассом. Если класс помечается как final, то ни один класс не может наследовать любую функцию из класса final.

### **Модификатор abstract** — используется для создания абстрактных классов и методов.

**Класс abstract** не может создать экземпляр. Если класс объявлен как abstract, то единственная цель для него быть расширенным. Класс не может быть одновременно abstract и final, так как класс final не может быть расширенным. Если класс содержит абстрактные методы, то он должен быть объявлен как abstract. В противном случае будет сгенерирована ошибка компиляции. Класс abstract может содержать как абстрактные методы, а также и обычные.

**Метод abstract** является методом, объявленным с любой реализацией. Тело метода (реализация) обеспечивается подклассом. Методы abstract никогда не могут быть final или strict. Любой класс, который расширяет абстрактный класс должен реализовать все абстрактные методы суперкласса, если подкласс не является абстрактным классом.Если класс в Java содержит один или несколько абстрактных методов, то класс должен быть объявлен как abstract. Абстрактный класс не обязан содержать абстрактные методы. Абстрактный метод заканчивается точкой с запятой. Пример: public abstract sample();

* **Модификатор synchronized** — используются в Java для потоков.

Ключевое слово synchronized используется для указания того, что метод может быть доступен только одним потоком одновременно. В Java модификатор synchronized может быть применен с любым из четырех модификаторов уровня доступа.

### Модификатор transient

### Переменная экземпляра, отмеченная как transient указывает виртуальной машине Java (JVM), чтобы пропустить определённую переменную при сериализации объекта, содержащего её. Этот модификатор включён в оператор, что создает переменную, предшествующего класса или типа данных переменной.

* **Модификатор volatile** — используются в Java для потоков.

В Java модификатор volatile используется, чтобы позволить знать JVM, что поток доступа к переменной всегда должен объединять свою собственную копию переменной с главной копией в памяти. Доступ к переменной volatile синхронизирует все кэшированные скопированные переменные в оперативной памяти. Volatile может быть применен только к переменным экземпляра, которые имеют тип объект или private. Ссылка на объект volatile может быть null.

# 3. Что такое конструктор? Как вы отличите конструктор от любого другого метода? Сколько конструкторов может быть в классе? Что такое конструктор по умолчанию, может ли в классе совсем не быть конструкторов? Объясните, что делает оператор this() в конструкторе?

Конструктор- метод имя которого совпадает с Именем класса. У него отсутствует возвращаемое значение. В классе всегда присутствует хотя бы 1 конструктор, если он явно не прописан, то пописывается неявно на этапе компиляции. Кол-во конструкторов не ограничено. Оператор this() вызывает конструктор аргументы которого прописаны в скобочках . В этом случае рекурсия недопустима, а так же должен быть конструктор который без this().

# 4. Опишите процедуру инициализации полей класса и полей экземпляра класса. Когда инициализируются поля класса, а когда – поля экземпляров класса. Какие значения присваиваются полям по умолчанию? Где еще в классе полям могут быть присвоены начальные значения?

Поля могут принадлежать классу или объекту. Поля, принадлежащие классу, являются статическими, а поля, принадлежащие объекту, - нестатическими. Статические поля доступны без создания объекта класса. Соответственно инициализироваться статические и нестатические поля должны в разное время: одни до создания объекта класса, а другие после.

|  |  |  |
| --- | --- | --- |
| Название | Применимость | Описание |
| Инициализация в месте объявления поля | Поля класса, поля объекта | Применяется, если инициализация может быть произведена коротким выражением и доступен контекст, необходимый для ее проведения |
| Инициализационный блок | Поля класса, поля объекта | Применяется, если инициализационный код неудобно записывать одним выражением или же, например, нужна обработка проверяемых исключений. В случае объектов может применяться для инициализации полей объектов анонимных классов. |
| Конструктор класса | Поля объекта | Применяется, если для инициализации нужны параметры конструктора |

Статические переменные инициализируются во время загрузки класса и в статическом блоке. Порядок выполнения сверху вниз. В отличии от полей класса, поля объекта инициализируются во время конструирования экземпляра класса. В Java существует 3 типа такой инициализации:

* инициализация в месте объявления
* инициализация в нестатическом блоке
* инициализация в конструкторе

## Инициализация в конструкторе и наследование

Выполним следующий код:

public class InheritanceInitOrder {

static class A {

String a;

A() {

a = "a";

System.out.println("a initialized");

System.out.println("b=" + ((B)this).b);

}

}

static class B extends A {

String b;

B() {

b = "b";

System.out.println("b initialized");

System.out.println("b=" + b);

}

}

public static void main(String[] args) throws ClassNotFoundException {

new B();

}

}

В out будет выведено:

a initialized

b=null

b initialized

b=b

Вывод свидетельствует о том, что инициализация выполнялась следующим образом:

* конструктор B первым делом вызвал конструктор предка - класса A
* конструктор A проинициализировал поле *a* объекта А
* при возврате из конструктора A, конструктор B проинициализировал поле *b* объекта B
* так же что компилятор перенес всю нестатическую инициализацию объекта в конструктор класса. При этом статический инициализационный блок остался без изменений.

# 5. Дайте определение перегрузке методов. Как вы думаете, чем удобна перегрузка методов? Укажите, какие методы могут перегружаться, и какими методами они могут быть перегружены? Можно ли перегрузить методы в базовом и производном классах? Можно ли private метод базового класса перегрузить public методов производного? Можно ли перегрузить конструкторы, и можно ли при перегрузке конструкторов менять атрибуты доступа у конструкторов?

Перегрузка метода заключается в следующем —создание метода с таким же именем, но с другим набором параметров. Например, в классе может быть несколько методов с названием summa, но с разным набором параметров. Здесь необходимо добавить важное замечание — имя параметра НЕ ИМЕЕТ значения. Т.е. если вы сделаете два метода summa с двумя параметрами типа double и с разными именами, это будет ошибкой.  Перегрузить можно любой метод, в том числе и конструктор. Нельзя переопределить частный метод, но вы можете ввести его в производный класс без проблем. Если попытаться применить аннотацию @Override, то будет ошибку времени компиляции

# 6. Объясните, что такое раннее и позднее связывание? Перегрузка – это раннее или позднее связывание? Объясните правила, которым следует компилятор при разрешении перегрузки; в том числе, если методы перегружаются примитивными типами, между которыми возможно неявное приведение или ссылочными типами, состоящими в иерархической связи.

Порядок поиска функции перегрузки если явно она не указана:

1. Расширение (widening).
2. Приведение к родительскому типу.
3. Упаковка (autoboxing and unboxing).
4. Аргументы переменной длины (varargs).

# 7. Объясните, как вы понимаете, что такое неявная ссылка this? В каких методах эта ссылка присутствует, а в каких – нет, и почему?

This – ссылка которая неявно передается в метод. Ссылается на объект который вызвал метод/поле => static не имеет её

# 8. Что такое финальные поля, какие поля можно объявить со спецификатором final? Где можно инициализировать финальные поля?

Для класса это означает, что класс не сможет иметь подклассов, т.е. запрещено наследование. Это полезно при создании immutable (неизменяемых) объектов. Следует также отметить, что к абстрактным классам (с ключевым словом abstarct), нельзя применить модификатор final, т.к. это взаимоисключающие понятия. Для метода final означает, что он не может быть переопределен в подклассах. Это полезно, когда мы хотим, чтобы исходную реализацию нельзя было переопределить. Для переменных примитивного типа это означает, что однажды присвоенное значение не может быть изменено. 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.

# 9. Что такое статические поля, статические финальные поля и статические методы. К чему имеют доступ статические методы? Можно ли перегрузить и переопределить статические методы? Наследуются ли статические методы?

Вы НЕ можете получить доступ к НЕ статическим членам класса, внутри статического контекста, как вариант, метода или блока.

В отличие от локальных переменных, статические поля и методы НЕ потокобезопасны (Thread-safe) в Java. Учитывая что каждый экземпляр класса имеет одну и ту же копию статической переменной, то такая переменная нуждается в защите — «залочивании» классом. Поэтому при использовании статических переменных, убедитесь, что они должным образом синхронизированы (synchronized), во избежание проблем, например таких как «состояние гонки» (race condition).

Статические методы имеют преимущество в применении, т.к. отсутствует необходимость каждый раз создавать новый объект для доступа к таким методам. Статический метод можно вызвать, используя тип класса, в котором эти методы описаны. Именно поэтому, подобные методы как нельзя лучше подходят в качестве методов-фабрик (factory), и методов-утилит (utility). Класс java.lang.Math — замечательный пример, в котором почти все методы статичны, по этой же причине классы-утилиты в Java финализированы (final).

Другим важным моментом является то, что вы НЕ можете переопределять (Override) статические методы. Если вы объявите такой же метод в классе-наследнике (subclass), т.е. метод с таким же именем и сигнатурой, вы лишь «спрячете» метод суперкласса (superclass) вместо переопределения. Это явление известно как сокрытие методов (hiding methods). Это означает, что при обращении к статическому методу, который объявлен как в родительском, так и в дочернем классе, во время компиляции всегда будет вызван метод исходя из типа переменной. В отличие от переопределения, такие методы не будут выполнены во время работы программы.

Модификатор static также может быть объявлен в статичном блоке, более известным как «Статический блок инициализации» (Static initializer block), который будет выполнен во время загрузки класса. Если вы не объявите такой блок, то Java соберёт все статические поля в один список и выполнит его во время загрузки класса. Однако, статичный блок НЕ может пробросить перехваченные исключения, но может выбросить не перехваченные. В таком случае возникнет «Exception Initializer Error». На практике, любое исключение возникшее во время выполнения и инициализации статических полей, будет завёрнуто Java в эту ошибку. Это также самая частая причина ошибки «No Class Def Found Error», т.к. класс не находился в памяти во время обращения к нему.

Полезно знать, что статические методы связываются во время компиляции, в отличие от связывания виртуальных или не статических методов, которые связываются во время исполнения на реальном объекте. Следовательно, статические методы не могут быть переопределены в Java, т.к. полиморфизм во время выполнения не распространяется на них. Это важное ограничение, которое необходимо учитывать, объявляя метод статическим. В этом есть смысл, только тогда, когда нет возможности или необходимости переопределения такого метода классами-наследниками. Методы-фабрики и методы-утилиты хорошие образцы применения модификатора static. Джошуа Блох выделил несколько преимуществ использования статичного метода-фабрики перед конструктором, в книге «[Effective Java](http://forcoder.ru/java/java-effektivnoe-programmirovanie-739)», которая является обязательной для прочтения каждым программистом данного языка.

Во время сериализации, также как и transient переменные, статические поля не сериализуются. Действительно, если сохранить любые данные в статическом поле, то после десериализации новый объект будет содержать его первичное (по-умолчанию) значение, например, если статическим полем была переменная типа int, то её значение после десериализации будет равно нулю, если типа float – 0.0, если типа Object – null. Честно говоря, это один из наиболее часто задаваемых вопросов касательно сериализации на собеседованиях по Java. Не храните наиболее важные данные об объекте в статическом поле!

И напоследок, поговорим о static import. Данный модификатор имеет много общего со стандартным оператором import, но в отличие от него позволяет импортировать один или все статические члены класса. При импортировании статических методов, к ним можно обращаться как будто они определены в этом же классе, аналогично при импортировании полей, мы можем получить доступ без указания имени класса. Данная возможность появилась в Java версии 1.5, и при должном использовании улучшает читабельность кода. Наиболее часто данная конструкция встречается в тестах *JUnit*, т.к. почти все разработчики тестов используют static import для assert методов, например assertEquals() и для их перегруженных дубликатов.

# 10. Что такое логические и статические блоки инициализации? Сколько их может быть в классе, в каком порядке они могут быть размещены и в каком порядке вызываются?

Логическим блоком называется код, заключенный в фигурные скобки и не принадлежащий ни одному методу текущего класса. Предназначен для первичной инициализации, но могут содержать вызовы методов и обращения к полям текущего класса. Операции с полями класса внутри логического блока до явного объявления этого поля возможны только при использовании ссылки this, представляющую собой ссылку на текущий объект. Логический блок может быть объявлен со спецификатором static. В этом случае он вызывается только один раз в жизненном цикле приложения при создании объекта или при обращении к статическому методу (полю) данного класса. Вызываются сверху вниз в любом кол-ве

# 11. Что представляют собой методы с переменным числом параметров, как передаются параметры в такие методы и что представляет собой такой параметр в методе? Как осуществляется выбор подходящего метода, при использовании перегрузки для методов с переменным числом параметров?

При возникновении конфликтов, связанных с тем, что нельзя определить какой из перегруженных методов будет вызван будет ошибка компиляции