Оглавление

[1. Чем является класс Object? Перечислите известные вам методы класса Object, укажите их назначение. 2](#_Toc21644877)

[2. Что такое хэш-значение? Объясните, почему два разных объекта могут сгенерировать одинаковые хэш-коды? 3](#_Toc21644878)

[3. Как вы думаете, для чего используется наследование классов в java-программе? Приведите пример наследования. Как вы думаете, поля и методы, помеченными модификатором доступа private, наследуются? 4](#_Toc21644879)

[4. Укажите, как вызываются конструкторы при создании объекта производного класса? Что в конструкторе класса делает оператор super()? Возможно ли в одном конструкторе использовать операторы super() и this()? 4](#_Toc21644880)

[5. Объясните, как вы понимаете утверждения: “ссылка базового класса может ссылаться на объекты своих производных типов” и “объект производного класса может быть использован везде, где ожидается объект его базового типа”. Верно ли обратное и почему? 4](#_Toc21644881)

[6. Что такое переопределение методов? Как вы думаете, зачем они нужны? Можно ли менять возвращаемый тип при переопределении методов? Можно ли менять атрибуты доступа при переопределении методов? Можно ли переопределить методы в рамках одного класса? 5](#_Toc21644882)

[7. Определите правило вызова переопределенных методов. Можно ли статические методы переопределить нестатическими и наоборот? 5](#_Toc21644883)

[8. Какие свойства имеют финальные методы и финальные классы? Как вы думаете, зачем их использовать? 5](#_Toc21644884)

[9. Укажите правила приведения типов при наследовании. Напишите примеры явного и неявного преобразования ссылочных типов. Объясните, какие ошибки могут возникать при явном преобразовании ссылочных типов. 5](#_Toc21644885)

[10. Что такое объект класса Class? Чем использование метода getClass() и последующего сравнения возвращенного значения с Type.class отличается от использования оператора instanceof? 5](#_Toc21644886)

[11. Укажите правила переопределения методов equals(), hashCode() и toString(). 5](#_Toc21644887)

[12. Что такое абстрактные классы и методы? Зачем они нужны? Бывают ли случаи, когда абстрактные методы содержат тело? Можно ли в абстрактных классах определять конструкторы? Могут ли абстрактные классы содержать неабстрактные методы? Можно ли от абстрактных классов создавать объекты и почему? 5](#_Toc21644888)

[13. Что такое интерфейсы? Как определить и реализовать интерфейс в java-программе? Укажите спецификаторы, которые приобретают методы и поля, определенные в интерфейсе. Можно ли описывать в интерфейсе конструкторы и создавать объекты? Можно ли создавать интерфейсные ссылки и если да, то на какие объекты они могут ссылаться? 6](#_Toc21644889)

[14. Для чего служит интерфейс Clonable? Как правильно переопределить метод clone() класса Object, для того, что объект мог создавать свои адекватные копии? 6](#_Toc21644890)

[15. Для чего служат интерфейсы Comparable и Comparator? В каких случаях предпочтительнее использовать первый, а когда – второй? Как их реализовать и использовать? 7](#_Toc21644891)

## Чем является класс Object? Перечислите известные вам методы класса Object, укажите их назначение.

Java Native Interface (JNI) — стандартный механизм для запуска кода, под управлением виртуальной машины Java (JVM), который написан на языках С/С++ или Ассемблера, и скомпонован в виде динамических библиотек, позволяет не использовать статическое связывание. Это даёт возможность вызова функции С/С++ из программы на Java, и наоборот.

Модификатор native сигнализирует о том, что метод реализован в платформо-зависимом коде, часто на языке С.

Этот модификатор может быть применен только к методам, но не классам и переменным.

Тело нативного метода должно заканчиваться на (;) как в абстрактных методах, идентифицируя то, что реализация опущена.

package java.lang;

public class Object {

private static native void registerNatives();

static {

registerNatives();

}

public final native Class<?> getClass();

public native int hashCode();

public boolean equals(Object obj) {

return (this == obj);

}

protected native Object clone() throws CloneNotSupportedException;

public String toString() {

return getClass().getName() + "@" + Integer.toHexString(hashCode());

}

public final native void notify();

public final native void notifyAll();

public final native void wait(long timeout) throws InterruptedException;

public final void wait(long timeout, int nanos) throws InterruptedException {

if (timeout < 0) {

throw new IllegalArgumentException("timeout value is negative");

}

if (nanos < 0 || nanos > 999999) {

throw new IllegalArgumentException( "nanosecond timeout value out of range");

}

if (nanos >= 500000 || (nanos != 0 && timeout == 0)) {

timeout++;

}

wait(timeout);

}

public final void wait() throws InterruptedException {

wait(0);

}

protected void finalize() throws Throwable { }

}

java.lang.Object особый класс, имеющий уникальный class file, в котором в качестве суперкласса не указан ни один класс. Создать такой же класс средствами языка Java нельзя, но также затруднительно, если вообще возможно, сделать это и манипуляциями с байтами class файла.

## Что такое хэш-значение? Объясните, почему два разных объекта могут сгенерировать одинаковые хэш-коды?

Хеш-функция ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) hash function от hash — «превращать в фарш», «мешанина»[[1]](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F#cite_note-_ff142c473b3a285a-1)), или функция свёртки — функция, осуществляющая преобразование [массива](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%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)) входных данных произвольной длины в (выходную) [битовую](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) строку установленной длины, выполняемое [определённым алгоритмом](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC). Преобразование, производимое хеш-функцией, называется хешированием. Исходные данные называются входным массивом, «ключом» или «сообщением». Результат преобразования (выходные данные) называется «[хешем](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%81%D1%83%D0%BC%D0%BC%D0%B0" \o "Хеш-сумма)», «хеш-кодом», «хеш-суммой», «сводкой [сообщения](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D0%B5)».

Хеш-функции применяются в следующих случаях:

* при построении [ассоциативных массивов](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2);
* при поиске дубликатов в сериях наборов данных;
* при построении уникальных идентификаторов для наборов данных;
* при вычислении [контрольных сумм](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D1%83%D0%BC%D0%BC%D0%B0) от данных (сигнала) для последующего обнаружения в них ошибок (возникших случайно или внесённых намеренно), возникающих при хранении и/или передаче данных;
* при сохранении паролей в системах защиты в виде хеш-кода (для восстановления пароля по хеш-коду требуется функция, являющаяся [обратной](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) по отношению к использованной хеш-функции);
* при выработке [электронной подписи](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C) (на практике часто подписывается не само сообщение, а его «хеш-образ»);
* и др.

В общем случае (согласно [принципу Дирихле](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%94%D0%B8%D1%80%D0%B8%D1%85%D0%BB%D0%B5_(%D0%BA%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%82%D0%BE%D1%80%D0%B8%D0%BA%D0%B0))) нет однозначного соответствия между хеш-кодом (выходными данными) и исходными (входными) данными. Возвращаемые хеш-функцией значения (выходные данные) менее разнообразны, чем значения входного массива (входные данные). Случай, при котором хеш-функция преобразует более чем один массив входных данных в одинаковые сводки, называется «[коллизией](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%BB%D0%B8%D0%B7%D0%B8%D1%8F_%D1%85%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8)». Вероятность возникновения коллизий используется для оценки качества хеш-функций.

## Как вы думаете, для чего используется наследование классов в java-программе? Приведите пример наследования. Как вы думаете, поля и методы, помеченными модификатором доступа private, наследуются?

С помощью наследования можно расширить функционал уже имеющихся классов за счет добавления нового функционала или изменения старого. Производный класс имеет доступ ко всем методам и полям базового класса (даже если базовый класс находится в другом пакете) кроме тех, которые определены с модификатором private. При этом производный класс также может добавлять свои поля и методы. Если в базовом классе определены конструкторы, то в конструкторе производного классы необходимо вызвать один из конструкторов базового класса с помощью ключевого слова super.

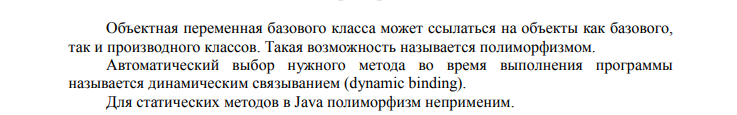
## Укажите, как вызываются конструкторы при создании объекта производного класса? Что в конструкторе класса делает оператор super()? Возможно ли в одном конструкторе использовать операторы super() и this()?

Сначала вызываются конструкторы базового класса, а затем конструкторы производного класса. Super() вызывает конструктор базового класса, не может одновременно использоваться вместе с this().

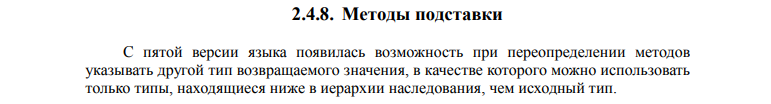
## Объясните, как вы понимаете утверждения: “ссылка базового класса может ссылаться на объекты своих производных типов” и “объект производного класса может быть использован везде, где ожидается объект его базового типа”. Верно ли обратное и почему?

## Что такое переопределение методов? Как вы думаете, зачем они нужны? Можно ли менять возвращаемый тип при переопределении методов? Можно ли менять атрибуты доступа при переопределении методов? Можно ли переопределить методы в рамках одного класса?









## Определите правило вызова переопределенных методов. Можно ли статические методы переопределить нестатическими и наоборот?

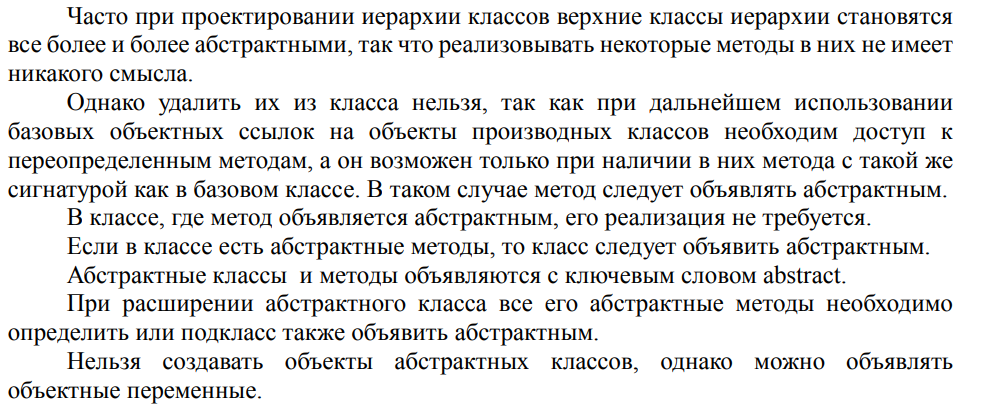
## Какие свойства имеют финальные методы и финальные классы? Как вы думаете, зачем их использовать?

## Укажите правила приведения типов при наследовании. Напишите примеры явного и неявного преобразования ссылочных типов. Объясните, какие ошибки могут возникать при явном преобразовании ссылочных типов.

## Что такое объект класса Class? Чем использование метода getClass() и последующего сравнения возвращенного значения с Type.class отличается от использования оператора instanceof?

## Укажите правила переопределения методов equals(), hashCode() и toString().

## Что такое абстрактные классы и методы? Зачем они нужны? Бывают ли случаи, когда абстрактные методы содержат тело? Можно ли в абстрактных классах определять конструкторы? Могут ли абстрактные классы содержать неабстрактные методы? Можно ли от абстрактных классов создавать объекты и почему?



**Абстрактный метод** – это метод без тела. Вы просто объявляете метод, не определяя его, с использованием ключевого слова abstract в объявлении метода. Все объявленные внутри интерфейса в языке Java методы – по умолчанию абстрактные.

Суть абстрактного класса заключается в том, что он не завершён, и его нужно завершить в классах-наследниках. То есть этот класс не готов к использованию. В нём, например, может отсутствовать реализация каких-то методов. Раз класс не готов к использованию, то нельзя создавать его объект. А вот экземпляры наследников абстрактного класса создавать можно.

## Что такое интерфейсы? Как определить и реализовать интерфейс в java-программе? Укажите спецификаторы, которые приобретают методы и поля, определенные в интерфейсе. Можно ли описывать в интерфейсе конструкторы и создавать объекты? Можно ли создавать интерфейсные ссылки и если да, то на какие объекты они могут ссылаться?

Интерфейс — это совокупность методов и правил взаимодействия элементов системы. Другими словами, интерфейс определяет как элементы будут взаимодействовать между собой. Интерфейсы применяются для добавления к классам новых возможностей и не может быть в базовых классах. Интерфейс только гарантирует какие методы должен выполнить класс, но как он их выполняет интерфейс не контролирует.

Поля интерфейса по умолчанию final static. Все методы открыты. Интерфейс считается реализованным если реализованы все методы интерфейса.

В интерфейсе могут быть методы по умолчанию, вложенные интерфейсы, поля.

Интерфейсные ссылки могут ссылаться только на те объекты, которые реализовали интерфейсы.

## Для чего служит интерфейс Clonable? Как правильно переопределить метод clone() класса Object, для того, что объект мог создавать свои адекватные копии?

Метод clone() объявлен в классе Object с сигнатурой native, чтобы обеспечить доступ к стандартному механизму "поверхностного копирования" объектов (копируются значения всех полей, включая ссылки на сторонние объекты); он объявлен, как protected, чтобы нельзя было вызвать этот метод у не переопределивших его объектов.

Интерфейс Cloneable является маркерным (не содержит объявлений методов). Он нужен только для обозначения самого факта, что данный объект готов к тому, чтобы быть клонированным. Вызов переопределённого метода clone() у не Cloneable объекта вызовет выбрасывание CloneNotSupportedException (это описано в объявлении метода).

Сам класс Object не является Cloneable, т.к. он является корневым классом в иерархии и это привело бы к тому, что все классы являются по умолчанию Cloneable.

Для реализации "глубокого клонирования" (когда вместо копирования ссылок на содержащиеся объекты создаются новые экземпляры) необходимо самим описать весь алгоритм действий в переопределённом методе clone().

Хорошей альтернативой сложной системе клонирования является "копирующий конструктор", принимающий, в качестве параметра, экземпляр собственного класса и создающий на его основе новый экземпляр.

## Для чего служат интерфейсы Comparable и Comparator? В каких случаях предпочтительнее использовать первый, а когда – второй? Как их реализовать и использовать?

Интерфейс Comparable

В интерфейсе Comparable объявлен только один метод compareTo (Object obj), предназначенный для упорядочивания объектов класса. Данный метод удобно использовать для сортировки списков или массивов объектов.

Метод compareTo (Object obj) сравнивает вызываемый объект с obj. В отличие от метода equals, который возвращает true или false, compareTo возвращает:

0, если значения равны;

Отрицательное значение (обычно -1), если вызываемый объект меньше obj;

Положительное значение (обычно +1), если вызываемый объект больше obj.

Если типы объектов не совместимы при сравнении, то compareTo (Object obj) может вызвать исключение ClassCastException. Необходимо помнить, что аргумент метода compareTo имеет тип сравниваемого объекта класса.

Обычные классы Byte, Short, Integer, Long, Double, Float, Character, String уже реализуют интерфейс Comparable.

Интерфейс Comparator : compare, compareTo

В интерфейсе Comparator объявлен метод compare (Object obj1, Object obj2), который позволяет сравнивать между собой два объекта. На выходе метод возвращает значение 0, если объекты равны, положительное значение или отрицательное значение, если объекты не тождественны.

Метод может вызвать исключение ClassCastException, если типы объектов не совместимы при сравнении.

Отличие интерфейсов Comparator и Comparable

Интерфейс Comparable используется только для сравнения объектов класса, в котором данный интерфейс реализован. Т.е. interface Comparable определяет логику сравнения объекта определенного ссылочного типа внутри своей реализации (по правилам разработчика).

Comparator представляет отдельную реализацию и ее можно использовать многократно и с различными классами. Т.е. interface Comparator позволяет создавать объекты, которые будут управлять процессом сравнения (например при сортировках).

Interface Comparable задает свойство сравнения объекту реализующему его. То есть делает объект сравнимым (по правилам разработчика).    
Interface Comparator позволяет создавать объекты, которые будут управлять процессом сравнения (например при сортировках).