**1 Что такое классы и объекты?**Ответ:

**Класс** – логическое описание чего-либо, шаблон, с помощью которого можно создавать реальные экземпляры этого самого чего-либо. Другими словами, это просто описание того, какими должны быть созданные сущности: какими свойствами и методами должны обладать.

Класс - определяет форму и сущность объекта и является логической конструкцией, на основе которой построен весь язык Java. Наиболее важная особенность класса состоит в том, что он определяет новый тип данных, которым можно воспользоваться для создания объектов этого типа, т.е. класс — это шаблон (чертеж), по которому создаются объекты (экземпляры класса). Для определения формы и сущности класса указываются данные, которые он должен содержать, а также код, воздействующий на эти данные.

Объект – это некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением, имеющая определённые свойства (атрибуты) и операции над ними (методы).

Создание объекта

Чтобы непосредственно использовать класс в программе, надо создать его объект. Процесс создания объекта двухступенчатый: вначале объявляется переменная данного класса, а затем с помощью ключевого слова new и конструктора непосредственно создается объект, на который и будет указывать объявленная переменная :

**2 Что такое инкапсуляция?**

Ответ:

Инкапсуляция - это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса). При использовании объектно-ориентированного подхода не принято применять прямой доступ к свойствам какого-либо класса из методов других классов.

Инкапсуляция связывает данные с манипулирующим ими кодом и позволяет управлять доступом к членам класса из отдельных частей программы, предоставляя доступ только с помощью определенного ряда методов, что позволяет предотвратить злоупотребление этими данными

**3 Какие модификаторы доступа сущ-т и зачем они нужны?**

Ответ:

Модификаторы доступа — это чаще всего ключевые слова, которые регулируют уровень доступа к разным частям твоего кода. Почему «чаще всего»? Потому что один из них установлен по умолчанию и не обозначается ключевым словом :) Всего в Java есть четыре модификатора доступа. Перечислим их в порядке от самых строгих до самых «мягких»:

private;

protected;

default (package visible);

public.

Модификаторы доступа private, protected, public ставятся перед именем класса, метода или поля и ограничивают доступ к нему. К локальным переменным модификаторы доступа не применимы.  
Помимо этих трех явных модификаторов, есть еще так называемый default-модификатор, или модификатор по умолчанию, иначе говоря – это отсутствие всякого модификатора.  Но это отсутствие тоже подразумевает свои правила доступа (видимость только внутри пакета).

**4 Как получить доступ к private полю объекта?**

Ответ:

**Private** — наиболее строгий модификатор доступа. Он ограничивает видимость данных и методов пределами одного класса.

Если член класса «private», то он будет доступен только внутри того же класса. Это наиболее ограниченный доступ, поэтому член класса не будет виден для внешнего мира. Обычно переменные класса и методы, предназначенные для использования только внутри класса, делают private.

**5 Что такое наследование и зачем оно нужно?**

Ответ:

Одним из основополагающих принципов ООП является наследование, который позволяет создать класс (суперкласс), определяющий какие-то общие черты набора классов, а затем этот общий класс может наследоваться другими, более специализированными классами (подклассами), каждый из которых будет добавлять свои особые характеристики. Подкласс наследует члены, определённые в суперклассе, добавляя к ним собственные.

**6 Что такое полиморфизм?**

Ответ:

Полиморфизм в [языках программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%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%D1%8F) и [теории типов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%82%D0%B8%D0%BF%D0%BE%D0%B2) — способность [функции](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%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)) обрабатывать данные разных [типов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Существует несколько разновидностей полиморфизма. Две принципиально различных из них были описаны [Кристофером Стрэчи](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D1%80%D1%8D%D1%87%D0%B8,_%D0%9A%D1%80%D0%B8%D1%81%D1%82%D0%BE%D1%84%D0%B5%D1%80&action=edit&redlink=1) в [1967 году](https://ru.wikipedia.org/wiki/1967_%D0%B3%D0%BE%D0%B4_%D0%B2_%D0%BD%D0%B0%D1%83%D0%BA%D0%B5): это [параметрический полиморфизм](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%BE%D0%BB%D0%B8%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC) и [ad-hoc-полиморфизм](https://ru.wikipedia.org/wiki/Ad-hoc-%D0%BF%D0%BE%D0%BB%D0%B8%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC" \o "Ad-hoc-полиморфизм), причём первая является истинной формой, а вторая — мнимой; прочие формы являются их подвидами или сочетаниями. Параметрический полиморфизм подразумевает исполнение одного и того же кода для всех допустимых типов аргументов, тогда как ad-hoc-полиморфизм подразумевает исполнение потенциально разного кода для каждого типа или подтипа аргумента. [Бьёрн Страуструп](https://ru.wikipedia.org/wiki/%D0%91%D1%8C%D1%91%D1%80%D0%BD_%D0%A1%D1%82%D1%80%D0%B0%D1%83%D1%81%D1%82%D1%80%D1%83%D0%BF" \o "Бьёрн Страуструп) определил полиморфизм как «один интерфейс — много реализаций», но это определение покрывает лишь ad-hoc-полиморфизм.

**7 Что такое конструкторы, зачем они нужны, как взаимодействуют конструкторы родителей и наследников?**

Ответ:

конструктор — это специальный метод, который вызывается при создании экземпляра объекта. Другими словами, когда вы используете новое ключевое слово. Цель конструктора — инициализировать вновь созданный объект перед его использованием.

Что такое конструктор в Java и зачем он нужен?

Рассмотрим два примера.

public class Car {

String model;

int maxSpeed;

public static void main(String[] args) {

Car bugatti = new Car();

bugatti.model = "Bugatti Veyron";

bugatti.maxSpeed = 407;

}

}

Мы создали наш автомобиль и установили для него модель и максимальную скорость. Однако в реальном проекте у объекта Car явно будет не 2 поля. А, например, 16 полей!

public class Car {

String model;//модель

int maxSpeed;//максимальная скорость

int wheels;//ширина дисков

double engineVolume;//объем двигателя

String color;//цвет

int yearOfIssue;//год выпуска

String ownerFirstName;//имя владельца

String ownerLastName;//фамилия владельца

long price;//цена

boolean isNew;//новая или нет

int placesInTheSalon;//число мест в салоне

String salonMaterial;//материал салона

boolean insurance;//застрахована ли

String manufacturerCountry;//страна-производитель

int trunkVolume;//объем багажника

int accelerationTo100km;//разгон до 100 км/час в секундах

public static void main(String[] args) {

Car bugatti = new Car();

bugatti.color = "blue";

bugatti.accelerationTo100km = 3;

bugatti.engineVolume = 6.3;

bugatti.manufacturerCountry = "Italy";

bugatti.ownerFirstName = "Amigo";

bugatti.yearOfIssue = 2016;

bugatti.insurance = true;

bugatti.price = 2000000;

bugatti.isNew = false;

bugatti.placesInTheSalon = 2;

bugatti.maxSpeed = 407;

bugatti.model = "Bugatti Veyron";

}

}

Мы создали новый объект Car. Одна проблема: полей-то у нас 16, а проинициализировали мы только 12! Попробуй сейчас по коду найти те, которые мы забыли! Не так-то просто, да? В такой ситуации программист может легко ошибиться и пропустить инициализацию какого-то поля. В итоге поведение программы станет ошибочным:

public class Car {

String model;//модель

int maxSpeed;//максимальная скорость

int wheels;//ширина дисков

double engineVolume;//объем двигателя

String color;//цвет

int yearOfIssue;//год выпуска

String ownerFirstName;//имя владельца

String ownerLastName;//фамилия владельца

long price;//цена

boolean isNew;//новая или нет

int placesInTheSalon;//число мест в салоне

String salonMaterial;//материал салона

boolean insurance;//застрахована ли

String manufacturerCountry;//страна-производитель

int trunkVolume;//объем багажника

int accelerationTo100km;//разгон до 100 км/час в секундах

public static void main(String[] args) {

Car bugatti = new Car();

bugatti.color = "blue";

bugatti.accelerationTo100km = 3;

bugatti.engineVolume = 6.3;

bugatti.manufacturerCountry = "Italy";

bugatti.ownerFirstName = "Amigo";

bugatti.yearOfIssue = 2016;

bugatti.insurance = true;

bugatti.price = 2000000;

bugatti.isNew = false;

bugatti.placesInTheSalon = 2;

bugatti.maxSpeed = 407;

bugatti.model = "Bugatti Veyron";

System.out.println("Модель Bugatti Veyron. Объем двигателя - " + bugatti.engineVolume + ", багажника - " + bugatti.trunkVolume + ", салон сделан из " + bugatti.salonMaterial +

", ширина дисков - " + bugatti.wheels + ". Была приоберетена в 2018 году господином " + bugatti.ownerLastName);

}

}

**8 Что такое абстрактные классы и методы?**

**абстрактный класс** — это максимально абстрактная, очень приблизительная «заготовка» для группы будущих классов. Эту заготовку нельзя использовать в готовом виде — слишком «сырая». Но она описывает некое общее состояние и поведение, которым будут обладать будущие классы — наследники абстрактного класса.

Абстрактные методы

Вы объявляете их, добавляя ключевое слово abstract перед объявлением метода:

public abstract class MyAbstractClass {

public abstract void abstractMethod();

}

Информация об абстрактных методах:

Не имеет реализации.

У него просто есть подпись метода. Так же, как методы в интерфейсе.

Если у класса есть абстрактный метод, весь класс должен быть объявлен как абстрактный.

Не все методы в абстрактном классе должны быть абстрактными, может иметь и неабстрактные.

Подклассы абстрактного класса должны реализовывать (переопределять) все абстрактные методы его абстрактного суперкласса. Неабстрактные методы суперкласса просто наследуются как есть. Они также могут быть отменены, если это необходимо.

**9 Что такое статические поля и методы?**

Ответ:

Модификатор static в Java напрямую связан с классом. Если поле статично, значит оно принадлежит классу, если метод статичный — аналогично: он принадлежит классу. Исходя из этого, можно обращаться к статическому методу или полю, используя имя класса. Например, если поле count статично в классе Counter, значит, вы можете обратиться к переменной запросом вида: Counter.count.

Поля данных, объявленные в классе как static, являются общими для всех объектов класса и называются переменными класса. Если один объект изменит значение такого поля, то это изменение увидят все объекты. Для работы со статическими атрибутами используются статические методы, объявленные со спецификатором static. Такие методы являются методами класса, не привязаны ни к какому объекту и не содержат указателя this на конкретный объект, вызвавший метод. Статические методы реализуют парадигму «раннего связыва­ния», жестко определяющую версию метода на этапе компиляции. По причине недоступности указателя this статические поля и методы не могут обращаться к нестатическим полям и методам напрямую, так как для обращения к статическим полям и методам достаточно имени класса, в котором они определены.

**10 От какого класса наследуются все классы?**

Ответ:

Класс, от которого наследуется дочерний класс, называется родительским классом (parent class), предком, суперклассом (superclass) или базовым классом (base class). В самой вершине иерархии наследования находится класс Object , от которого наследуются все классы, для которых не указан явно суперкласс.

**11 Что такое cast и instanceof ?**

Ответ:

способ приведения объектов с помощью методовClass:

public void whenDowncastToCatWithCastMethod\_thenMeowIsCalled() {

Animal animal = new Cat();

if (Cat.class.isInstance(animal)) {

Cat cat = Cat.class.cast(animal);

cat.meow();

}

}

Мы часто используем оператор instanceof перед понижающим преобразованием, чтобы проверить, принадлежит ли объект определенному типу:

**12 Где и как хранятся объекты в памяти?**

Ответ:

В Java все объекты динамически размещаются в Heap. Это отличается от C ++, где объектам может быть выделена память либо в стеке, либо в куче. В C ++, когда мы размещаем объект с помощью new (), объект размещается в Heap, в противном случае в стеке, если не глобальный или статический.

В Java, когда мы объявляем только переменную типа класса, создается только ссылка (память не выделяется для объекта). Чтобы выделить память для объекта, мы должны использовать new (). Таким образом, объекту всегда выделяется память в куче

**13 Есть ли в джава множественное наследование?**

Ответ:

Множественное наследование – способность создавать классы с множеством классов-родителей. В отличии от других популярных объектно-ориентированных языков, вроде С++, язык Java не поддерживает множественное наследование классов. Не поддерживает он его из-за вероятности столкнуться с «проблемой алмаза» и вместо этого предпочитает обеспечивать некий комплексный подход для его решения, используя лучшие варианты из тех, которыми мы можем достичь аналогичный результат наследования.

**14 Что такое интерфейс?**

Ответ:

Интерфейс — это «проводник» между человеком и программой, операционной системой, техническим устройством или способ взаимодействия приложений между собой. Человек дает команды с помощью интерфейса, устройство их анализирует и отвечает. Основные задачи, для решения которых он предназначен:

ввод и отображение информации (звук, изображение);

управление отдельными приложениями;

обмен данными с другими устройствами;

взаимодействие с операционной системой.

Интерфейс подразумевает взаимодействие не только человека и техники, но и компьютер-программа, программа-программа, компьютер-устройство. Например, когда устройства подключают к системному блоку компьютера, как способ взаимодействия используют разъем.

## Виды интерфейсов

Одни виды взаимодействия позволяют получить больше контроля над компьютером или смартфоном, но требуют дополнительных навыков. Другие — более комфортные, но предоставляют меньше возможностей. У каждого типа есть свои особенности.

### Командная строка

Через командную строку можно выполнить максимальное количество операций — это прямой способ общения с операционной системой. Чтобы набрать команду, нужно ввести текст на языке компьютера и нажать Enter, компьютер начнет выполнять.

### Графический и текстовый

Графика упрощает взаимодействие с компьютером, с ней работать гораздо легче и комфортнее, чем с текстом. В роли графического интерфейса выступают такие элементы:

иконки;

меню;

списки;

рисунки и схемы;

другие графические элементы.

Например, при взаимодействии с Windows используют иконки и окна, для ввода подключают мышь. На смартфоне устройством ввода служит сенсорный дисплей.

С помощью ключевого слова interface можно полностью абстрагировать интерфейс класса от его реализации, то есть указать, **что** именно должен выполнять класс, но не **как** это делать. Синтаксически интерфейсы аналогичны классам, но не содержат переменные экземпляра, а объявления их методов, как правило, не содержат тело метода. Каждый интерфейс может быть реализован любым количеством классов. Кроме того, один класс может реализовать любое количество интерфейсов. Чтобы реализовать интерфейс, в классе должен быть переопределён весь набор методов интерфейса