**Класс** – это шаблон для объекта; шаблон, описывающий общие свойства группы объектов.

**Объект** - экземпляр класса.

У каждого объекта есть **свойства и методы** (действия, которые может делать объект)**.**

Class object = new Class(); -> ИмяКласса ИмяОбъекта = НовыйЭкземплярКласса

**Наследование** (англ. inheritance) — это механизм, позволяющий **создавать классы на основе другого класса**.

Благодаря наследованию, можно не прописывать все методы и свойства класса заново. Можно просто добавить недостающие методы и параметры.

Класс, являющийся основой, называют: базовым, супер, **родительским**.

Класс, который создают, называют: потомок, **наследник** или производный класс.

Наследники **имеют доступ ко всем методам и переменным родителя** (кроме тех, у который модификатор доступа что private).

Для того, чтобы унаследовать класс, нужно использовать ключевое слово **extends**.

Java не поддерживает наследование нескольких классов. **Один класс - один родитель.**

**Нельзя наследовать самого себя!**

**Полиморфизм** — это возможность применения одноименных методов с одинаковыми или различными наборами параметров в одном классе или в группе классов, связанных отношением наследования.

Смысл заключается в том, что можно давать **одно и то же название для методов, которые имеют одинаковый смысл**, но принимают разные типы данных.

Полиморфизм — это **принцип** (Один интерфейс - много методов).

**Способы реализации: перегрузка, переопределение**

**Инкапсуляция** позволяет регулировать уровни доступа в программе.

С помощью инкапсуляции мы **защищаем** данные от неправомерного использования.

Применение:

* **Модификаторы доступа**
* **Геттеры и Сеттеры**

**Модификаторы доступа** помогают регулировать **кто может видеть** элементы кода, и **кто может их менять**.

Модификаторы доступа применяются к полям (т.е. переменным), к методам, в геттерах.

Если **не указывать** модификатор, то ставится модификатор **default**.

Типы:

* **public** – виден отовсюду, можно обращаться из любой части программы
* **default** – виден только в границах пакета (package)
* **private** – виден только в границах класса
* **protected** – виден в границах пакета и для наследников

Модификаторы пишутся перед названиями переменных, методов и классов

Все четыре модификатора применяются к методам и полям (переменным).

К классам могут применяться все четыре модификатора только если они вложенные. В противном случае, применим только public и default.

**Конструкторы** — это специальные методы, которые вызывается при создании объекта. Они "конструируют" новый объект определенного класса.

**Конструкторы инициализируют объекты.**

Конструктор всегда **присутствует в классе по умолчанию**, просто он явно не указан.

Явно прописывая конструктор, **можно регулировать**, какие параметры и в каком количестве нужно задать для создания объекта определенного класса. (имя, цвет, возраст и т.д.)

Цель модификатора **final** - сделать **дальнейшее изменение** объекта **невозможным**.

Суть в том, что первое заданное значение меняться не будет.

Переменные с final — это **константы**. При этом их принято писать заглавными буквами

Методы и final: **при наследовании данный метод нельзя переопределить.**

Классы и final: **нельзя создать наследников этого класса (методы тоже становятся final)**

Цель модификатора **static** - сделать **переменную или метод "независимыми"** от объекта.

Если **метод (переменная)** имеет модификатор static, то это значит, что она **объявляется для класса**. Без этого модификатора - для объектов.

Статические методы/переменные **можно вызвать без создания объекта класса**.

**Переменная** с модификатором static **одна для всех объектов класса**, не статическая переменная "создается заново" для каждого объекта.

**Все константы являются статическими.** Они объявляются с использованием сочетания static final.

**Метод вызывается без создания объекта класса.**

К статическим методам **можно обращаться и через название объекта, и название класса**. К нестатическим методам нужно обращаться исключительно через название объектов класса.

**Статические методы нельзя переопределять.**

Статический метод не может ссылаться на нестатическую переменную.

**Переопределение** (override) — это способ **поддержания полиморфизма** в Java. Это то, что мы можем создать в классе-наследнике класс с таким же названием, как и класс, который мы унаследовали от родителя,

Для того, чтобы **переопределить** (изменить) метод класса-родителя, пишем над ним **@Override (аннотация).**

Если создать наследника, например, класса Cat (который наследуется от Animal), то он будет видеть переопределенный метод класса Cat, а не родительский Animal.

Когда мы наследуем какой-либо класс, мы наследуем и все его методы. Но если нам хочется изменить какой-либо из методов, который мы наследуем, мы можем всего-навсего переопределить его. Мы не обязаны, например, создавать отдельный метод с похожим названием для наших нужд, а унаследованный метод будет "мертвым грузом" лежать в нашем классе.

**Перегрузка** методов — это **возможность создавать несколько методов с одинаковым названием, но разными параметрами**.

**Если создать два метода с одинаковым названием, которые будут требовать одни и те же параметры, возникнет ошибка.**

Называть методы одинаково — это очень удобно. Например, если у нас есть метод, который ищет корень квадратный из числа, гораздо легче запомнить одно название (например, sqrt()), чем по одному отдельному названию на этот же метод, написанный для каждого типа (int, short, double).

**Не стоит путать переопределение и перегрузку методов**. И то и то позволяет нам создать разные методы с одинаковыми названиями.

**Но:**

* Переопределение происходит, когда "первая строчка" метода (сигнатура) у исходного метода и у "нового" метода одинаковая. То-есть и название, и возвращаемый тип, и принимаемый тип не меняются.
* Если же что-то из этого меняется (т.е. меняется сигнатура) - мы уже имеем дело с перегрузкой.

**Геттеры и Сеттеры** - общепринятый способ вводить данные ("set") или получать данные ("get")

Геттеров и сеттеры защищают содержимое программы, когда ей пользуется кто-то другой.

private String name;

private String color;

public String getName(){

return name;

} - метод, который возвращает переменную типа String и при этом не требует никаких данных.

public void setName(String a){

name = a;

} - метод ничего не возвращает (void), но при этом требует String - новое значение для переменной name.

**РАЗНОЕ**

Если нужно изменить метод класса-родителя чуть-чуть (например, дописать пару строк), то можно вызвать родительский метод с помощью ключевого слова **super**. (сначала вызов родительского метода, а затем то, что дописали)

Для обозначения переменной, объявленной для всего класса, используется слово **this.**

class Cat{

private String color;

public void setColor(String color){

**this.color** = color;

}

}

Метод **main** - не должен присутствовать в каждом классе. Он **необходим для запуска программы** - поэтому понадобиться только один раз.

В **public static void main** используется модификатор доступа public, который означает, что **содержание main "видно отовсюду"** (метод, который запускает всю программу, не может быть приватным)