

1. Конспект урока

1. Что такое объектно-ориентированное программирование

В мире существует достаточно много парадигм программирования. Различные парадигмы содержат в себе наборы идей и методологий по работе в коде. Например, в **функциональном** программировании всё является функцией, то есть аргументами одних функций являются другие, более простые функции. Функциональное программирование пытается разделить данные и поведение, а **объектно-ориентированное программирование** объединяет эти концепции.

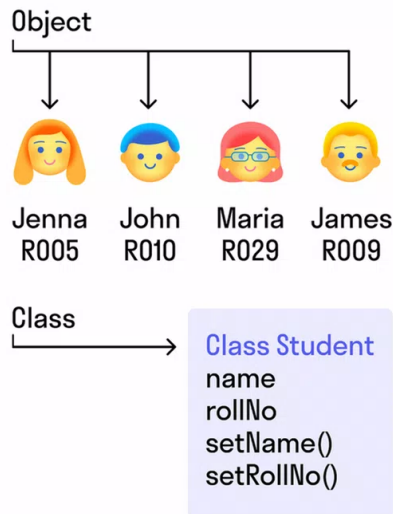
Java является объектно-ориентированным языком.

📌 **Объектно-ориентированное программирование** (в дальнейшем ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

ООП подразумевает построение приложений в виде множества различных объектов (каждый из них является реализацией какого-либо класса), которые взаимодействуют друг с другом. Причем любой объект может реализовывать не просто какой-то конкретный класс, а целую иерархию унаследованных друг от друга классов, каждый из которых внес частичку себя в общее целое.

Класс —

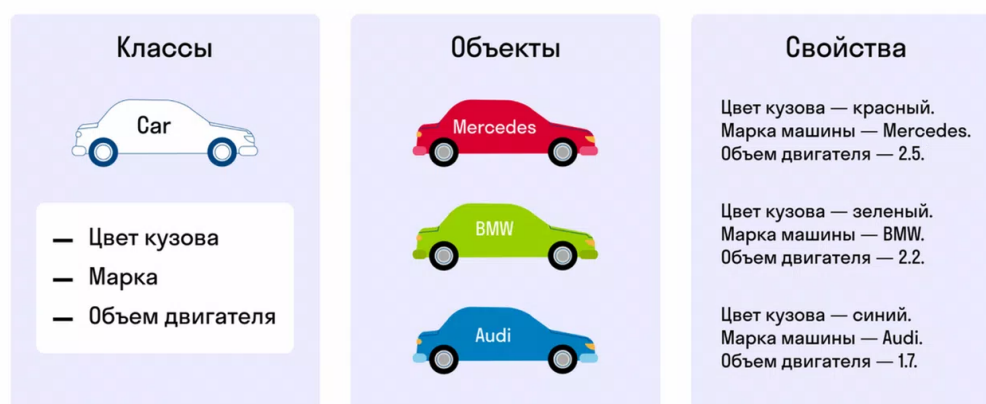
это чертеж, по которому
можно создавать объекты.



Похожее происходит в реальном мире. Например, вам нужно купить хлеб.

✎ Вы (объект-человек) надеваете одежду (объект-свитер, объект-джинсы), берете деньги (объект-кошелек вам в помощь), идете (вызываете метод «идти» объекта-человека) в магазин (объект-магазин), покупаете (обращаетесь к объекту-кошельку, берете у него объект-деньги, передаете его объекту-кассе/продавцу), получаете хлеб (объект-хлеб).

Пример классов



У объектов могут быть свойства.
Свойства — это переменные,
привязанные к объекту.

Объект Dog



Свойства объекта:

- String name
- int age
- int kids

Переменные,
в которых содержатся
данные об объекте



Как видите, всё происходит относительно реалистично (с поправкой на виртуальный мир и его логику). Такой «реалистичный» подход очень полезен при проектировании сложных систем, где множество составных частей и нужно наладить их взаимодействие между собой.


Чтобы создать Java-класс, необходимо нажать правой кнопкой мыши на папку (пакет), в которой планируется создание класса, и выбрать New → Java Class. После чего будет создан новый класс.

Для создания нового объекта необходимо использовать оператор new при инициализации переменной определенного класса. Например:

```
Car car = new Car();  
// ИмяКласса имяОбъекта = new ИмяКласса  
// Обратите внимание на именование: название класса всегда с большой буквы без
```

2. Принципы ООП

В центре ООП находится понятие объекта.

 **Объект** — это сущность, экземпляр класса, которой можно посылать сообщения и которая может на них реагировать, используя свои данные.

Данные объекта скрыты от остальной программы. Скрытие данных называется **инкапсуляцией**.

1 Инкапсуляция — это свойство системы, позволяющее объединить данные и методы, работающие с ними в классе, и скрыть детали реализации от пользователя.

Наличие инкапсуляции недостаточно объектной ориентированности языка — для этого требуется наличие **наследования**.


2 Наследование — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствуемой функциональностью. Класс, от которого производится наследование, называется **базовым, родительским** или **суперклассом**. Новый класс — **потомком, наследником** или **производным классом**.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован **полиморфизм**, то есть возможность объектов с одинаковой спецификацией иметь различную реализацию.

3 Полиморфизм — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре

объекта.

Существует мнение, что в ООП стоит выделять еще одну немаловажную характеристику — **абстракцию**. Официально ее не вносили в обязательные черты ООП, но списывать ее со счетов не стоит.


 **Абстрагирование** — это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, абстракция — это набор всех таких характеристик.

Принципы инкапсуляции, наследования, полиморфизма широко используются в языке Java. Также Java представляет собой набор объектов — экземпляров классов, которые в свою очередь создают иерархию. Всё это вместе делает Java ярким представителем ООП. А с тремя парадигмами ООП мы подробнее познакомимся на следующих уроках. А сейчас мы повторим, как создаются объекты.

3. Что такое конструктор?

Как мы определили с вами выше, ООП строится вокруг объектов. Но объекты не возникают сами собой, для их создания всегда используется конструктор.

Что же это такое?

 Конструктор инициализирует объект непосредственно во время его создания.

При создании объекта то, что пишется после ключевого слова `new`, и есть конструктор:

```
Car myCar = new Car();
```

Добавим конструктор в класс Car сразу после переменных. Имя конструктора совпадает с именем класса, в котором он используется, а синтаксис аналогичен синтаксису метода. Конструкторы не имеют возвращаемого типа. Так как в данном случае конструктор не может возвращать тип, отличный от текущего класса. В конструкторе мы устанавливаем значение переменным, которые необходимы для создания объекта нашего класса:

```
// Объявляем класс
public class Car {

    // объявляем переменные
        String brand;
        String color;
        double engine;

    // далее как раз объявляем конструктор, в его теле присваиваем значения
    Car() {
        System.out.println("Конструирование объекта Car");
        brand = "BMW";
        color = "black";
        engine = 4.4;
    }
}
```

Конструктор по умолчанию

Мы пока не определяли конструктор класса Car, но всё же мы имели возможность создавать объекты, так как если конструктор класса не определен явно, то в Java для класса создается *конструктор по умолчанию*.



Конструктор по умолчанию инициализирует все переменные экземпляра устанавливаемыми по умолчанию значениями.

Но в случае создания нашего собственного конструктора конструктор по умолчанию больше не используется. Класс с конструктором по умолчанию выглядит так:

```
public class Car {  
    String brand;  
    String color;  
    double engine;  
  
    Car() {  
    }  
    ...  
}
```

Конструктор с параметрами

При создании объектов зачастую необходимо сразу присваивать им конкретные значения полей, для этого используется **конструктор с параметрами**. Такие конструкторы называются параметризованными.

В следующем примере мы создаем конструктор с параметрами, на вход которого передаем три значения для инициализации трех переменных класса:

```
public class Car {  
    String brand;  
    String color;  
    double engine;  
  
    /**  
     * Конструктор класса Car  
     *  
     * @param brand - марка  
     * @param color - цвет  
     * @param engine - объем двигателя  
     */  
    Box(String brand, String color, double engine) {  
        this.brand = brand;  
        this.color = color;  
        this.engine = engine;  
    }  
}
```

Теперь в конструктор необходимо передать марку, цвет и объем двигателя нашего автомобиля. Этот конструктор можно использовать для установки необходимых значений прямо во время создания объекта, что наиболее корректно, так как объект не может существовать без определенных свойств и ожидать, когда ему их присвоят; эти свойства нужны сразу при создании объекта. Теперь мы не можем создавать объект класса `Car`, используя конструктор по умолчанию, так как определили свой собственный:

```
public class CarTest {  
    public static void main(String[] args) {  
        Car myCar1 = new Car("BMW", "black", 4.4);  
        Car myCar2 = new Car("Renault", "blue", 1.4);  
    }  
}
```

Код конструктора отвечает только за создание объекта. Не стоит вызывать из конструктора другие методы, так как метод может быть переопределен в подклассе и изменить инициализацию объекта.

Давайте также немного освежим в памяти темы, которые были нами изучены в конце первого курса и которые мы затрагивали в этом уроке.

4. Ключевое слово `this`

В данном уроке использовалось ключевое слово

`this`. Напомним, что



`this` является переменной, которая всегда ссылается на ваш конкретный объект.

В случае вызова в конструкторе мы, по сути, говорим, что нужно вызвать другой конструктор этого же класса, но с другими параметрами.

В случае применения конструкции вида

`this.someValue = someValue` ключевое слово `this` используется для того, чтобы явно показать, к какой конкретно переменной мы обращаемся.

5. Методы

Также мы затронули различные аспекты, которые касаются работы с методами.

Метод является поведением конкретного объекта или реализацией функционала, работающего идентично для каждого экземпляра класса, то есть является статическим. Если обобщить, то:



Метод — блок кода, который выполняет определенную функцию и позволяет переиспользовать себя в нескольких местах без необходимости снова писать один и тот же код.

Каждый метод должен отвечать за решение одной определенной задачи. Метод может возвращать какое-то значение определенного типа или же может быть помечен модификатором `void`, который означает, что метод только совершает какое-то действие, ничего при этом не возвращая.