**Вопросы 💎**

1. Чем отличается класс от объекта? Приведите примеры и того, и другого, отличные от указанных в уроке.

Объект (object) — это набор свойств, каждое из которых состоит из названия и значения.

А класс – это только схема (шаблон) создания и работы какого-то объекта. То есть класс определяет все свойства, которые характеризуют группу объектов.

Например, класс может описывать множество всех объектов (экземпляров данного класса).

объект: «мой смартфон», его свойствами могут быть марка, модель, диагональ, цвет корпуса и иные характеристики.

* *Класс* определяет все свойства (учитывая методы и все поля в Java, или свойства в C++), которые характеризуют группу объектов. Класс это абстрактная вещь, а не какой-либо конкретный член множества объектов, которые он описывает. Например, класс Employee может описывать множество всех сотрудников.
* *Экземпляр*, это воплощение класса в виде конкретного объекта. Например, Victoria может быть экземпляром класса Employee, представляющий собой конкретного сотрудника. Экземпляр класса имеет ровно столько свойств, сколько и родительский класс (не больше и не меньше).

1. Приведите примеры объектно-ориентированного подхода в реальной жизни.

*Возможный вариант ответа: например, библиотека могла бы содержать классы Книга (название, автор, год выпуска, количество страниц), Сотрудники (ФИО, должность, зарплата, дата начала работы) и Читатели (ФИО, контакты, номер читательского билета).*

Примеры объектно-ориентированного подхода в реальной жизни:

Таксопарк: классы Машины (марки, модели, пробег, класс, цвета и пр.), Водители (ФИО, стаж, данные водительского удостоверения), Пассажиры (ФИО, контакты, история поездок).

Приведите примеры объектно-ориентированного подхода в программировании.

*Возможный вариант ответа: например, товары в интернет магазине (название товара, тип, цена), комментарии в соцсети (автор комментария, дата, текст комментария), видео в Youtube (название видео, его адрес, автор и дата загрузки).*

Примеры объектно-ориентированного подхода в программировании:

Список работников на портале организации (ФИО, наименование отдела, должность, стаж, контактные данные, адрес офиса).

1. Что такое конструктор? Самостоятельно изучите и напишите, какие бывают виды конструкторов.

Конструктор — это специальный метод, который вызывается при создании объекта с использованием класса. То есть это функция, которая используется, чтобы создавать однотипные объекты. Когда создаётся новый объект с помощью оператора new (экземпляр класса), в этот момент вызывается constructor.

По соглашениям конструкторы вызывают с помощью ключевого слова new, а также называют с большой буквы, причём обычно не глаголом, а существительным.

1. Что выведет код? Почему именно так?

function bike() {

    console.log(this.name)

}

var name = 'ninja'

var obj1 = {name: "pomidor", bike: bike}

var obj2 = {name: "site", bike: bike}

bike()             //ninja

obj1.bike()        //pomidor

obj2.bike()        //site

//bike здесь - это функция - конструктор, через которую создаются объекты (метод, инициализирующий объекты) и их свойство name выводится в консоль.

//this — это ссылка на некий объект, к свойствам которого можно получить доступ внутри вызова функции

1. Чем статические свойства и методы отличаются от нестатических? В каких ситуациях они применяются?

Статические методы применяются к самому классу, а не к отдельным объектам.

1. Самостоятельно изучите, что такое геттеры и сеттеры, приведите пример класса с их использованием.

Геттеры и сеттеры относятся к *свойствам-аксессорам (accessor properties)*. По своей сути это функции, которые используются для присвоения и получения значения, но во внешнем коде они выглядят как обычные свойства объекта.

Геттер - функция чтения свойства (используется для получения значения свойства, возвращает значение или undefined).

Сеттер - функция записи свойства (используется для установки значения свойства. Принимает единственным аргументом новое значение, присваиваемое свойству.

Например, у нас есть объект user со свойствами name и surname:

let user = {

    name: "Serseya",

    surname: "Lanister"

};

А теперь добавим свойство объекта fullName для полного имени, которое в нашем случае "John Smith". Само собой, мы не хотим дублировать уже имеющуюся информацию, так что реализуем его при помощи аксессора:

let user = {

    name: " Serseya ",

    surname: " Lanister ",

    get fullName() {

        return `${this.name} ${this.surname}`;

    }

};

alert(user.fullName); // Serseya Lanister

Снаружи свойство-аксессор выглядит как обычное свойство. В этом и заключается смысл свойств-аксессоров. Мы не *вызываем* user.fullName как функцию, а *читаем* как обычное свойство: геттер выполнит всю работу за кулисами.

На данный момент у FullName есть только геттер. Если мы попытаемся назначить user.FullName=, произойдёт ошибка:

let user = {

get fullName() {

return `...`;

}

};

user.fullName = "Тест"; // Ошибка (у свойства есть только геттер)

Давайте исправим это, добавив сеттер для user.fullName:

let user = {

name: "John",

surname: "Smith",

get fullName() {

return `${this.name} ${this.surname}`;

},

set fullName(value) {

[this.name, this.surname] = value.split(" ");

}

};

// set fullName запустится с данным значением

user.fullName = "Alice Cooper";

alert(user.name); // Alice

alert(user.surname); // Cooper

В итоге мы получили «виртуальное» свойство fullName. Его можно прочитать и изменить.

Сеттеры могут понадобиться, например, для модификации значения при записи свойств. В примере ниже мы модифицируем дату и записываем в нужном формате.

const updatedAt = {

get date() {

return this.\_date

},

set date(value) {

this.\_date = new Intl.DateTimeFormat('en-US').format(value)

}

}

Запишем дату и время в поле date:

updatedAt.date = new Date(2030, 11, 12)

console.log(updatedAt.date)

И получим дату в нужном формате: 12/12/2030.

1. Что выведет код? Почему именно так?

class Person {

    constructor(name) {

    this.name = name;

    }

}

const member = new Person("John")

console.log(typeof member)               //object

// Person - конструктор (класс), member - объект, который создали на основе существующего класса Person с параметром (name="John"). Поэтому выводится тип объекта member - object

1. Что выведет код? Почему именно так?

const person = {

    name: "Valera",

    age: 23

}

let city = person.city

city = "Amsterdam"

console.log(person)

// {name: 'Valera', age: 23}

// age: 23

// name: "Valera"

// свойство city со значением "Amsterdam" не попало в объект, так как запись неверная. Верно так:

const person = {

    name: "Valera",

    age: 23

}

person.city = "Amsterdam"

console.log(person)