**Контрольные вопросы**

**1. В чем заключается суть ООП программирования и его отличие от традиционного процедурного программирования?**

Ключевое различие между ООП и процедурным программированием заключается в том, что основное внимание процедурного программирования уделяется разбиению задачи программирования на набор переменных и подпрограмм, в то время как ООП фокусируется на разбиении задачи программирования на объекты, которые инкапсулируют данные. и методы.

**2. Что понимается под Классом и Объектом в JS?**

Класс — это шаблон, описывающий свойства и методы, которыми будет обладать любой объект, созданный на основе этого класса.

Конкретным воплощением этого представления, класса является объект.

**3. Пример синтаксиса класса в JS**

class Person{ }

**4. Как осуществляется создание объектов класса?**

const tom = new Person();

**5. Как происходит обращение к полям и методам внутри класса?**

Обращение к полям класса:

class Person{

name;

age;

}

const tom = new Person();

tom.name = "Tom";

tom.age = 37;

Обращение к методам класса:

class Person{

name;

age;

move(place){

console.log(`Go to ${place}`);

}

eat(){

console.log("Eat apples");

}

}

const tom = new Person();

tom.move("Hospital"); // Go to Hospital

tom.move("Cinema"); // Go to Cinema

tom.eat();

**6. Как обозначаются в JS публичные, приватные и защищённые поля и методы?**

Поля без префиксов являются публичными.

Названия приватных полей и методов предваряется символом #.

Защищённые свойства обычно начинаются с префикса \_.

**7. Для чего в ООП в JS используется служебное слово static?**

Оно обозначает, что данный метод или поле является статическим.

class StaticMethodCall {

static staticMethod() {

return 'Вызван статический метод';

}

static anotherStaticMethod() {

return this.staticMethod() + ' из другого статического метода';

}

}

StaticMethodCall.staticMethod();

// 'Вызван статический метод'

StaticMethodCall.anotherStaticMethod();

// 'Вызван статический метод из другого статического метода'

**8. Что такое геттеры и сеттеры ? Для чего и как они используются в ООП в JS?**

Оба метода - get и set имеют одинаковые названия. Как правило, они опосредуют доступ к некоторому приватному полю. Метод set предназначен для установки. Он принимает в качестве параметра новое значение. Далее в методе set мы можем выполнить ряд действий при установке.

Метод get предназначен для получения значения. Здесь мы можем определить какую-нибудь логику при возвращении значения.

let obj = {

get propName() {

// геттер, срабатывает при чтении obj.propName

},

set propName(value) {

// сеттер, срабатывает при записи obj.propName = value

}

};

**9. Что понимается под базовыми принципами ООП: инкапсуляция, наследование и полиморфизм?**

Инкапсуляция — способ спрятать сложную логику внутри класса, предоставив программисту лаконичный и понятный интерфейс для взаимодействия с сущностью.

Наследование — способ легко и просто расширить существующий класс, дополнив его функциональностью.

Полиморфизм — принцип «один интерфейс — множество реализаций». Например, метод print может вывести текст на экран, распечатать его на бумаге или вовсе записать в файл.

**10. На каждый принцип подберите в сети Интернет примеры реализации базовых принципов ООП на ЯП JS: инкапсуляция, наследование, полиморфизм.**

Наследование:

**class** **Enemy** {

**constructor**(power) {

this.power = power

}

attack = () => console.log(`I'm attacking with a power of ${this.power}!`)

}

**class** **Alien** **extends** **Enemy** {

**constructor** (name, phrase, power) {

super(power)

this.name = name

this.phrase = phrase

this.species = "alien"

}

fly = () => console.log("Zzzzzziiiiiinnnnnggggg!!")

sayPhrase = () => console.log(this.phrase)

}

Инкапсуляция:

**class** **Alien** **extends** **Enemy** {

#birthYear // Сначала нужно объявить закрытое свойство, используя в начале его имени символ '#'

**constructor** (name, phrase, power, speed, birthYear) {

super(name, phrase, power, speed)

this.species = "alien"

this.#birthYear = birthYear // Затем внутри функции конструктора мы присваиваем его значение

}

fly = () => console.log("Zzzzzziiiiiinnnnnggggg!!")

howOld = () => console.log(`I was born in ${this.#birthYear}`) // и используем его в соответствующем методе

}

// Привычным образом выполняем инстанцирование

**const** alien1 = **new** Alien("Ali", "I'm Ali the alien!", 10, 50, 10000)

Полиморфизм:

**class** **Enemy** **extends** **Character** {

**constructor**(name, phrase, power, speed) {

super(speed)

this.name = name

this.phrase = phrase

this.power = power

}

sayPhrase = () => console.log(this.phrase)

attack = () => console.log(`I'm attacking with a power of ${this.power}!`)

}

**class** **Alien** **extends** **Enemy** {

**constructor** (name, phrase, power, speed) {

super(name, phrase, power, speed)

this.species = "alien"

}

fly = () => console.log("Zzzzzziiiiiinnnnnggggg!!")

attack = () => console.log("Now I'm doing a different thing, HA!") // Переопределение родительского метода

}

**const** alien1 = **new** Alien("Ali", "I'm Ali the alien!", 10, 50)

alien1.attack() // вывод: "Now I'm doing a different thing, HA!"