**Урок 10**

**1. Об'єктно-орієнтоване програмування**

**1.1. Процедурне програмування**

Програма написана в процедурному стилі очікує на вхід якісь дані, виконує ряд якихось процедур, вони ж називаються функціями і після цього повертає якийсь результат. Результат обчислень. Відразу розглянемо з прикладу:

const width = 5;

        const height = 10;

        function calcRectArea(width, height){

            return width \* height;

        }

        calcRectArea(width, height)

у нас є 2 змінні це ширина та висота. Є функція (процедура) яка ширину і висоту приймає аргументами і за підсумком ця функція повертає результат множення ширини на висоту, тобто площа прямокутника. Підсумок викликавши цю функцію з якимись даними ми можемо отримати площу прямокутника. Подібний підхід і називається процедурним

Але коли програми стали більшими стало зрозуміло, що такий підхід не підходить. Адже стало складно керувати програмами.

Тому і був придуманий об'єктно орієнтований підхід.

**1.2.** **Сутності ООП**

У контексті ООП ці характеристики ім'я, прізвище називаються властивостями, а дії які може здійснювати той чи інший об'єкт називаються методами (наприклад ходити, малювати, говорити).

Давайте відразу розглянемо на прикладі:

**class Rectangle {**

**width;**

**height;**

**constructor(w, h){**

**this.width = w;**

**this.height = h;**

**}**

**calcArea(){**

**return this.width \* this.height;**

**}**

**}**

**const rect = new Rectangle(5,10)**

**const rect2 = new Rectangle(52,102)**

**const rect3 = new Rectangle(10,102)**

**rect.calcArea()**

Є клас прямокутник у нього є дві властивості – це ширина і висота. Так само є один єдиний метод, за допомогою якого ми можемо порахувати площу цього прямокутника. під цим в даному випадку мається на увазі об'єкт у якого буде викликаний цей метод.

Також у будь-якого класу є конструктор. Це спеціальний метод, блок інструкцій який викликається при створенні об'єкта. Він так само може приймати деякі аргументи. Зазвичай у конструкторі властивостям об'єкта надаються якісь значення. У разі аргументами ми приймаємо висоту і ширину і зберігаємо їх.

За допомогою оператора new ми можемо створити об'єкт. Об'єкт це окремий екземпляр якогось класу. При цьому аргументами ми передаємо ширину та висоту прямокутника. У створеного об'єкта ми можемо викликати відповідний метод, який поверне нам площу об'єкта, у якого ми викликали цей метод.

При цьому з будь-якого класу ми можемо створити стільки об'єктів, скільки нам потрібно. У даному прикладі кожен з прямокутників має різну висоту і ширину.

Кожний клас може включати стільки властивостей і методів скільки буде потрібно. Але гарною практикою є робити класи під окремі завдання.

**1.3.** **Класс**

Своїми словами клас це деякий опис показників. Наприклад людина, яка має ім'я, прізвище, по батькові, вік, вага. Тобто властивості якими ми можемо охарактеризувати людину. Ще раз це деякий список характеристик, за допомогою якого ми можемо охарактеризувати людину.

У контексті ООП подібна характеристика називається класом. Тобто клас Людина, клас human, клас person.

**1.4.** **Екземпляр**

Конкретний представник класу, екземпляр називається об'єктом. Можна сказати що це вася пупкін якому 27 років. Клас це пошук характеристик, а об'єкт це конкретний екземпляр у якого кожна характеристика має якесь значення, наприклад ім'я одно Вася.

**1.5.** **Інтерфейс**

В об'єктно-орієнтованому програмуванні властивості та методи розділені на 2 групи:

Внутрішній інтерфейс - методи та властивості, доступні з інших методів класу, але не зовні класу.

Зовнішній інтерфейс - методи та властивості, доступні зовні класу.

Давайте розберемо це на прикладі, кавоварки. Проста зовні: кнопка, екран, кілька отворів… І, звичайно, результат – чудова кава! Але в середині

Безліч деталей. Але ми можемо користуватися нею, нічого про це не знаючи.

Кавоварки досить надійні, чи не так? Ми можемо користуватися ними роками, і якщо щось піде не так - віднесемо у ремонт.

Секрет надійності та простоти кавоварки – всі деталі добре налагоджені та заховані всередині.

Якщо ми знімемо захисний кожух з кавоварки, то користуватися нею буде набагато складніше (куди натискати?) і небезпечніше (може призвести до ураження електричним струмом).

Як побачимо, у програмуванні об'єкти схожі на кавоварки.

Але щоб приховати внутрішні деталі, ми використовуватимемо не захисний кожух, а спеціальний синтаксис мови та угоди.

**Тобто якщо казати простими словами:**

\* Клас — це шаблон за яким будуть створені однотипні об'єкти

\* Примірник/екземпляр - це об'єкт створений за класом

\* Інтерфейс - набір доступних методів та властивостей

**2. Прототипне наслідування**

**У програмуванні ми часто хочемо щось і розширити.**

**Наприклад, у нас є об'єкт user зі своїми властивостями та методами, і ми хочемо створити об'єкти admin та guest як його трохи змінені варіанти. Ми хотіли б повторно використовувати те, що має об'єкт user, не копіювати/перевизначати його методи, а просто створити новий об'єкт на його основі.**

**Прототипне успадкування – це можливість мови, яка допомагає у цьому.**

**У JavaScript об'єкти мають спеціальну приховану властивість [[Prototype]] (так воно названо у специфікації), яке або дорівнює null, або посилається на інший об'єкт. Цей об'єкт називається "прототип":**

**Прототип дає нам трохи магії. Коли ми хочемо прочитати властивість із object, а вона відсутня, JavaScript автоматично бере її з прототипу. У програмуванні такий механізм називається "прототипним успадкуванням". Багато цікаві можливості мови та техніки програмування ґрунтуються на ньому.**

**Властивість [[Prototype]] є внутрішньою та прихованою, але є багато способів задати його.**

**Одним із них є використання \_\_proto\_\_, наприклад так:**

        const planet = {

            pi: 3.1415926,

            calcSquare(){

                const{pi, radius} = this

                return 4\*pi\*radius\*\*2

            }

        }

        const earth = {

            name:'Earth',

            radius: 40000,

        }

        earth.\_\_proto\_\_ = planet

        const mars = {

            name:'Mars',

            radius: 20000

        }

        mars.\_\_proto\_\_ = planet

        console.log(earth.calcSquare())

console.log(mars.calcSquare())

**Тут рядок (\*) встановлює animal як прототип для rabbit.**

**Потім, коли alert намагається прочитати властивість rabbit.eats (\*\*), його немає в rabbit, тому JavaScript слідує за посиланням [[Prototype]] і знаходить його в animal**

**Тут ми можемо сказати, що "animal є прототипом rabbit" або "rabbit прототипно успадковує від animal".**

**Так що якщо у animal багато корисних властивостей та методів, то вони автоматично стають доступними у rabbit. Такі властивості називаються «успадкованими».**

**2.2. Метод hasOwnProperty()**

Метод hasOwnProperty() повертає логічне значення, що вказує, чи містить об'єкт зазначену властивість. Щоб викликати цей метод, вам потрібно використовувати змінну об'єкт, що містить, і передати ім'я властивості як параметр (це обов'язково). Метод hasOwnProperty() повертає логічне значення true, якщо об'єкт має властивість із зазначеним ім'ям і воно не успадковано. Якщо немає властивостей, які відповідають імені, яке ви передали як аргумент, або ця властивість успадкована від прототипу об'єкта, метод поверне false.

        console.log(earth.hasOwnProperty('pi'))

        console.log(earth.hasOwnProperty('calcSquare'))

**3. Класи**

Синтаксис літералу об'єкта дозволяє створити один об'єкт. Проте, часто потрібно створити багато однотипних об'єктів з однаковим набором властивостей, але різними значеннями та методами для взаємодії з ними. Все це потрібно робити динамічно, під час виконання програми. З цією метою використовують класи - спеціальний синтаксис оголошення функції для створення об'єктів.

**3.1.**  **Оголошення класу**

Оголошення класів починаються з ключового слова class за яким слідує ім'я класу. Інша частина синтаксису виглядає подібною до визначення методів в літералах об'єктів, не вимагаючи коми між ними. Наприклад, ось просте оголошення класу:

   class User {

        constructor(name, email){

        this.name = name;

        this.email = email;

        }

        }

        const user = new User('Ivan','Ivan@gmail.com');

        console.log(user);

        const user2 = new User('Lesya','Lesya@gmail.com');

        console.log(user2);

**3.2.**  **Конструктор класу**

Коли створюється екземпляр классу першим викликається вбудований в клас метод constructor. В який передаються аргументи.

class user{

constructor(name, email) {

this.name = name;

this.email = email

}

**3.3.**  **Об'єкт параметрів**

Клас може приймати велику кількість вхідних даних для властивостей майбутнього об'єкта. Тому, до них також можна застосувати патерн «Об'єкт параметрів», передаючи один об'єкт з логічно іменованими властивостями, замість непов'язаного набору аргументів.

class User{

// еквівалент конструктора user

constructor({name, email}) {

this.name = name;

this.email = email

}

}

const atreydos = new User({ name: 'Agamemnon', email: 'Agamemnon@gmail' });

console.log(atreydos)

const atreydos2 = new User({ name: 'Elrud', email: 'Elrud@gmail' });

console.log(atreydos2)

**3.4.** **Методи класу**

Для роботи з властивостями майбутнього екземпляра використовуються методи класу - функції, які будуть доступні екземпляру в його прототипі.

class User{

        constructor({name, email}) {

                this.name = name;

                this.email = email

            }

            updateName(newName){

                if(newName){

                    this.name = newName

                    return true

                }

                    return false

            }

        }

        const atreydos = new User({ name: 'Agamemnon', email: 'Agamemnon@gmail' });

        console.log(atreydos.name)

        atreydos.updateName('')

        console.log(atreydos.name)

        atreydos.updateName('Вориан Атрейдес')

        console.log(atreydos.name)

Ми зробили метод updateName(), який оновлює імя користувача але лише тоді коли в новому імї є хочаб одна літера. Перша консоль виводимо початкове імя, потім пробуємо оновити імя пустою строкою за допомогою методу updateName(). Але коли в друге виводимо імя в консоль бачимо що імя не змінилось так як в нас є вомога хочаб одна літера. Далі знову оновлюємо імя але вже передаємо реальну строку. В третій консолі виводимо нове імя.

**3.5.** **Приватні властивості**

Інкапсуляція - це концепція, що дозволяє приховати внутрішні деталі класу. Користувач класу повинен отримувати доступ тільки до публічного інтерфейсу - набору публічних властивостей і методів класу.

В класах інкапсуляція реалізується приватними властивостями, доступ до яких можна отримати тільки всередині класу.

    class User{

            #email;

        constructor({name, email}) {

                this.name = name;

                this.#email = email

            }

            updateName(newName){

                if(newName){

                    this.name = newName

                    return true

                }

                    return false

            }

            getEmail(){

                return this.#email

            }

        }

        const atreydos = new User({ name: 'Agamemnon', email: 'Agamemnon@gmail' });

        console.log(atreydos.email)

        console.log(atreydos.getEmail())

        // atreydos.updateName('')

        // console.log(atreydos.name)

        // atreydos.updateName('Вориан Атрейдес')

        // console.log(atreydos.name)

В даному прикладі в класі ми зробили приватне свойство Email. Якщо вивести в консоль atraydes.email ми бачимо андефайнд тому що ця властивість приватна. Щоб отримати до неї доступ ми маємо створити метод getEmail(). Та викликати в консолі цей метод.

**3.6.** **Геттери і сеттери**

class User{

            #email;

        constructor({name, email}) {

                this.name = name;

                this.#email = email

            }

            updateName(newName){

                if(newName){

                    this.name = newName

                    return true

                }

                    return false

            }

            // getEmail(){

            //     return this.#email

            // }

            get email(){

                return this.#email

            }

            set email(newEmail){

                if(newEmail.includes('@')){

                    this.#email = newEmail

                    return true

                }

                return false

            }

        }

        const atreydos = new User({ name: 'Agamemnon', email: 'Agamemnon@gmail' });

        console.log(atreydos.email)

        atreydos.email = 'Vorian@gmail.com'

        console.log(atreydos.email)

        // console.log(atreydos.getEmail())

        // atreydos.updateName('')

        // console.log(atreydos.name)

        // atreydos.updateName('Вориан Атрейдес')

        // console.log(atreydos.name)

**Щоб не використовувати методи для отримання або зміни приватних властивостей використовують сетори та гетори. Ключові слова сет и гет поставлені перед методами класу дозволяють працювати з ними як з просто властивостями, а не викликаючи їх як методи.**

**3.7.** **Статичні властивості**

Крім публічних і приватних властивостей майбутнього екземпляра, в класі можна оголосити його власні властивості, доступні тільки класові, але не його екземплярам - статичні властивості static. Вони корисні для зберігання інформації, що стосується класу.

        class User{

            static Roles = {

            MANAGER: "meneger",

            USER: "user",

            };

            #email;

            #role;

        constructor({name, email, role}) {

                this.name = name;

                this.#email = email

                this.#role = role

            }

            updateName(newName){

                if(newName){

                    this.name = newName

                    return true

                }

                    return false

            }

            // getEmail(){

            //     return this.#email

            // }

            // get email(){

            //     return this.#email

            // }

            // set email(newEmail){

            //     if(newEmail.includes('@')){

            //         this.#email = newEmail

            //         return true

            //     }

            //     return false

            // }

            get role() {

            return this.#role;

            }

            set role(newRole) {

            this.#role = newRole;

            }

            }

            const user = new User({

            email: "Agamemnon@gmail",

            role: User.Roles.MANAGER,

            });

        // const atreydos = new User({ name: 'Agamemnon', email: 'Agamemnon@gmail' });

        console.log(user.Roles); // undefined

        console.log(User.Roles); // {NAME: 'Вориан Атрейдес', POSITION: 'The only son Agamemnon '}

        console.log(user.role); // Вориан Атрейдес

        user.role = User.Roles.USER;

        console.log(user.role); // The only son Agamemnon

        // console.log(atreydos.email)

        // atreydos.email = 'Vorian@gmail.com'

        // console.log(atreydos.email)

        // console.log(atreydos.getEmail())

        // atreydos.updateName('')

        // console.log(atreydos.name)

        // atreydos.updateName('Вориан Атрейдес')

        // console.log(atreydos.name)

Статичні властивості оголошуються в тілі класу. Перед ім'ям властивості додається ключове слово static.

**3.8.** **Статичні методи**

У класі можна оголосити не тільки методи майбутнього екземпляра, а також методи, доступні тільки класу - статичні методи, які можуть бути як публічні, так і приватні. Синтаксис оголошення аналогічний статичним властивостям, за винятком того, що значенням буде метод.

class User{

            static Roles = {

            MANAGER: "manager",

            USER: "user",

            };

            static isValidRole(role){

                const roles = Object.values(User)

                 return roles.includes(role);

            }

            #email;

            #role;

        constructor({name, email, role}) {

                this.name = name;

                this.#email = email

                this.#role = role

            }

            updateName(newName){

                if(newName){

                    this.name = newName

                    return true

                }

                    return false

            }

            // getEmail(){

            //     return this.#email

            // }

            // get email(){

            //     return this.#email

            // }

            // set email(newEmail){

            //     if(newEmail.includes('@')){

            //         this.#email = newEmail

            //         return true

            //     }

            //     return false

            // }

            get role() {

            return this.#role;

            }

            set role(newRole) {

            this.#role = newRole;

            }

            }

            const user = new User({

            email: "Agamemnon@gmail",

            role: User.Roles.MANAGER,

            });

            if(User.isValidRole('admin')){

                user.role = 'admin'

            }

            console.log(user.role)

        // const atreydos = new User({ name: 'Agamemnon', email: 'Agamemnon@gmail' });

        // console.log(user.Roles); // undefined

        // console.log(User.Roles); // {NAME: 'Вориан Атрейдес', POSITION: 'The only son Agamemnon '}

        // console.log(user.role); // Вориан Атрейдес

        // user.role = User.Roles.USER;

        // console.log(user.role); // The only son Agamemnon

        // console.log(atreydos.email)

        // atreydos.email = 'Vorian@gmail.com'

        // console.log(atreydos.email)

        // console.log(atreydos.getEmail())

        // atreydos.updateName('')

        // console.log(atreydos.name)

        // atreydos.updateName('Вориан Атрейдес')

        // console.log(atreydos.name)

**Ми створили статичний метод чи є роль яку хочуть додати юзеру у списку допустимих.**

**3.9.** **Наслідування класів, Конструктор дочірнього класу**

Ключове слово extends дозволяє реалізувати наслідування класів, коли один клас (дочірній, похідний) наслідує властивості і методи іншого класу (батьківського).

class User{

            static Roles = {

            MANAGER: "manager",

            USER: "user",

            };

            static isValidRole(role){

                const roles = Object.values(User)

                 return roles.includes(role);

            }

            #email;

            #role;

        constructor({name, email, role}) {

                this.name = name;

                this.#email = email

                this.#role = role

            }

            updateName(newName){

                if(newName){

                    this.name = newName

                    return true

                }

                    return false

            }

            // getEmail(){

            //     return this.#email

            // }

            // get email(){

            //     return this.#email

            // }

            // set email(newEmail){

            //     if(newEmail.includes('@')){

            //         this.#email = newEmail

            //         return true

            //     }

            //     return false

            // }

            get role() {

            return this.#role;

            }

            set role(newRole) {

            this.#role = newRole;

            }

            }

            class Student extends User{

                constructor({name, email, role, university, scores}){

                    super({name, email, role});

                    this.university = university

                    this.scores = [...scores]

                }

            }

            const student = new Student({

                name:'Agamemnon',

                email: "Agamemnon@gmail",

                role: User.Roles.MANAGER,

                university: "Oxford",

                scores:[100, 90, 100, 80]

            })

            console.log(student)

            // const user = new User({

            // email: "Agamemnon@gmail",

            // role: User.Roles.MANAGER,

            // });

            // if(User.isValidRole('admin')){

            //     user.role = 'admin'

            // }

            // console.log(user.role)

        // const atreydos = new User({ name: 'Agamemnon', email: 'Agamemnon@gmail' });

        // console.log(user.Roles); // undefined

        // console.log(User.Roles); // {NAME: 'Вориан Атрейдес', POSITION: 'The only son Agamemnon '}

        // console.log(user.role); // Вориан Атрейдес

        // user.role = User.Roles.USER;

        // console.log(user.role); // The only son Agamemnon

        // console.log(atreydos.email)

        // atreydos.email = 'Vorian@gmail.com'

        // console.log(atreydos.email)

        // console.log(atreydos.getEmail())

        // atreydos.updateName('')

        // console.log(atreydos.name)

        // atreydos.updateName('Вориан Атрейдес')

        // console.log(atreydos.name)

**Ми створили дочірній клас student з двома додатковими властивостями: university and scores. В конструкторі дочірного класу обовязково спочатку викликати конструктор батьківського класу. Це робиться через ключове слово Super, Куди ми передаємо всі аргументи для батьківського класу.**

**3.11.** **Методи дочірнього класу**

В дочірньому класі можна оголошувати методи, які будуть доступні тільки його екземплярам.

class User{

            static Roles = {

            MANAGER: "manager",

            USER: "user",

            };

            static isValidRole(role){

                const roles = Object.values(User)

                 return roles.includes(role);

            }

            #email;

            #role;

        constructor({name, email, role}) {

                this.name = name;

                this.#email = email

                this.#role = role

            }

            updateName(newName){

                if(newName){

                    this.name = newName

                    return true

                }

                    return false

            }

            // getEmail(){

            //     return this.#email

            // }

            // get email(){

            //     return this.#email

            // }

            // set email(newEmail){

            //     if(newEmail.includes('@')){

            //         this.#email = newEmail

            //         return true

            //     }

            //     return false

            // }

            get role() {

            return this.#role;

            }

            set role(newRole) {

            this.#role = newRole;

            }

            }

            class Student extends User{

                constructor({name, email, role, university, scores}){

                    super({name, email, role});

                    this.university = university

                    this.scores = [...scores]

                }

                getAvarageScore(){

                    const sum = this.scores.reduce((acum, item)=> acum + item,0)

                    return sum / this.scores.length

                }

            }

            const student = new Student({

                name:'Agamemnon',

                email: "Agamemnon@gmail",

                role: User.Roles.MANAGER,

                university: "Oxford",

                scores:[100, 90, 100, 80]

            })

            console.log(student)

            console.log(student.getAvarageScore())

            // const user = new User({

            // email: "Agamemnon@gmail",

            // role: User.Roles.MANAGER,

            // });

            // if(User.isValidRole('admin')){

            //     user.role = 'admin'

            // }

            // console.log(user.role)

        // const atreydos = new User({ name: 'Agamemnon', email: 'Agamemnon@gmail' });

        // console.log(user.Roles); // undefined

        // console.log(User.Roles); // {NAME: 'Вориан Атрейдес', POSITION: 'The only son Agamemnon '}

        // console.log(user.role); // Вориан Атрейдес

        // user.role = User.Roles.USER;

        // console.log(user.role); // The only son Agamemnon

        // console.log(atreydos.email)

        // atreydos.email = 'Vorian@gmail.com'

        // console.log(atreydos.email)

        // console.log(atreydos.getEmail())

        // atreydos.updateName('')

        // console.log(atreydos.name)

        // atreydos.updateName('Вориан Атрейдес')

        // console.log(atreydos.name)

**У класі student ми створили метод getAvarageScore який вираховує середній бал. Цей метод доступний лише для обєктів створених за допомогою класу Student.**