**Лабораторна робота № 4  
Рефакторинг програмного коду**

Виконав студент групи **КН-19-2 Ільченко Дмитро**

**Мета роботи:** : навчитися виконувати реорганізацію програмного коду на підставі шаблонів рефакторінгу.

**Хід виконання роботи**

1. Вивчити теоретичні відомості.

2. Виконати аналіз програмного коду розроблюваного ПО і модульних тестів з метою виявлення погано організованого код.

3. Використовуючи шаблони рефакторінгу, виконати реорганізацію програмного коду розроблюваного ПО і модульних тестів.

4. Перевірити успішність виконання всіх модульних тестів.

5. Виконати опис вироблених операцій рефакторінгу (було-стало-шаблон рефакторінгу).

6. В разі необхідності коректувати проектну документацію (діаграми класів,

послідовностей).

7. Зробити висновки результатами виконання роботи.

Рефакторинг:

1. Метод **getAllTimeExercises** класу CourseBase повертав загальну кількість хвилин, яку користувач витратить для проходження курсу тренувань.

**Проблеми**:

* метод враховував тільки час самих вправ, а час на перерву – ні.

**Рішення:**

* Вирішена проблема **Rename Method** ( Назва методу не розкриває суть того, що він робить) – змінив назву методу
* Вирішена проблема **Extract Method** (вас є фрагмент коду, який можна згрупувати.) – Видалив цей фрагмент в новий метод (чи функцію) і викликаю його замість старого коду.
* Додавання додаткової логіки для обчислення

Було:

    public getAllTimeExercises(): number {

        let totalExecutonTime =  this.\_dataCourse.exercises.reduce((acc,item)=>{

            return acc + item.getData.ExecutionTime

         },0);

        return totalExecutonTime

    }

Стало:

    public getAllTimeExercisesWithBreakes(): number {

        let totalExecutonTime = this.\_dataCourse.exercises.reduce((acc, item) => {

            return acc + item.getData.ExecutionTime;

        }, 0);

        return this.calculateAllTimeExercisesWithBreaks(totalExecutonTime);

    }

    private calculateAllTimeExercisesWithBreaks(inputTotalExecutonTime: number): number {

        return Math.round((this.\_dataCourse.exercises.length \* 120 + (this.\_dataCourse.exercises.length - 1) \* 30 - inputTotalExecutonTime) / 60);

    }

1. Клас **CourseBase** має об’єкт з даними. Змінна **isCreateByUser** (тип даних: **boolean**) даного об’єкта повертала true або false у тому випадку чи створений даний курс користувачем, чи додатком.

**Проблеми**:

* Змінна дає розуміння ким створений курс тренувань, але якщо користувач створив курс, то нам потрібно знати його id для відображення саме йому його курс, а не всім користувачам.

**Рішення:**

* Було додано додаткове поле для id в тому випадку, якщо курс створений користувачем.

**Було :**

Поля об’єкту

export type dataCurrentCourse = {

    id: string;

    name: string;

    muscleZone: string;

    exercises: Array<Exercise>;

    lvlDifficulty: number;

    lastTimeExecution: string | Date;

    isCreateByUser: boolean;

    isUserFollow: boolean;

    image?: string;

};

Створення курсу додатком:

const ABSCourseBeginner = new CourseBase({

        name: "ABS Beginner",

        muscleZone: "Abs",

        exercises: allExercises.getExercises.filter((item) => item.getData.id.includes("abs-beginner")) || [],

        lvlDifficulty: 1,

        id: "efd92720-431a-11ed-8958-5f188e7ae6b8",

        lastTimeExecution: new Date().toLocaleString(),

        isCreateByUser: false

        isUserFollow: false,

        image: absImg,

    });

Створення курсу користувачем:

new CourseBase({

                id: uuidv4(),

                exercises: arrayCurrentExercise,

                isCreateByUser: true

                isUserFollow: false,

                lastTimeExecution: new Date().toLocaleString(),

                lvlDifficulty: Exercises.averageLvlDiffuculty(arrayCurrentExercise),

                muscleZone: muscleTypeInput.value.toLowerCase(),

                name: nameCustomCourseInput.value,

                image: customImg,

            });

**Стало:**  
Поля об’єкту

export type dataCurrentCourse = {

    id: string;

    name: string;

    muscleZone: string;

    exercises: Array<Exercise>;

    lvlDifficulty: number;

    lastTimeExecution: string | Date;

    isCreateByUser: checkIsCreateByUser;

    isUserFollow: boolean;

    image?: string;

};

export type checkIsCreateByUser ={

    state: boolean;

    userId: string;

}

Створення курсу додатком:

    const ABSCourseBeginner = new CourseBase({

        name: "ABS Beginner",

        muscleZone: "Abs",

        exercises: allExercises.getExercises.filter((item) => item.getData.id.includes("abs-beginner")) || [],

        lvlDifficulty: 1,

        id: "efd92720-431a-11ed-8958-5f188e7ae6b8",

        lastTimeExecution: new Date().toLocaleString(),

        isCreateByUser: { state: false, userId: null },

        isUserFollow: false,

        image: absImg,

    });

Створення курсу користувачем:

  let createdCourse = new CourseBase({

                id: uuidv4(),

                exercises: arrayCurrentExercise,

                isCreateByUser: { state: true, userId: usersManager.getctiveUser.about.id },

                isUserFollow: false,

                lastTimeExecution: new Date().toLocaleString(),

                lvlDifficulty: Exercises.averageLvlDiffuculty(arrayCurrentExercise),

                muscleZone: muscleTypeInput.value.toLowerCase(),

                name: nameCustomCourseInput.value,

                image: customImg,

            });

1. Метод **determineRatioOfPFC** класу **Calculating** розраховує співвідношення білків, жирів і вуглеводів, виходячи з добового споживання калорій та мети користувача (втратити вагу, підтримувати вагу, набрати вагу)

**Проблеми**:

* **Duplicate Code** – дублювання коду.(Фрагменти даного метода виглядають майже однаковими)

**Рішення:**

* Необхідно застосувати відокремлення методу, після чого викликати код створеного методу з тих ділянок, де він потрібен.

**Було:**

    public static determineRatioOfPFC(dailyCalorieIntake: number, goalUser: EnumGoalUser): Nutrients {

        let \_ratio: Nutrients = {

            dailyProtein: 0,

            dailyFat: 0,

            dailyCarbs: 0,

        };

        switch (goalUser) {

            case EnumGoalUser.LoseWeight: {

                \_ratio.dailyProtein = Math.round((dailyCalorieIntake \* 0.3) / this.PercentRatio.PROTEIN);

                \_ratio.dailyFat = Math.round((dailyCalorieIntake \* 0.2) / this.PercentRatio.FAT);

                \_ratio.dailyCarbs = Math.round((dailyCalorieIntake \* 0.5) / this.PercentRatio.GARBS);

                return \_ratio;

            }

            case EnumGoalUser.MaintainWeight: {

                \_ratio.dailyProtein = Math.round((dailyCalorieIntake \* 0.3) / this.PercentRatio.PROTEIN);

                \_ratio.dailyFat = Math.round((dailyCalorieIntake \* 0.3) / this.PercentRatio.FAT);

                \_ratio.dailyCarbs = Math.round((dailyCalorieIntake \* 0.4) / this.PercentRatio.GARBS);

                return \_ratio;

            }

            case EnumGoalUser.GainWeight: {

                \_ratio.dailyProtein = Math.round((dailyCalorieIntake \* 0.35) / this.PercentRatio.PROTEIN);

                \_ratio.dailyFat = Math.round((dailyCalorieIntake \* 0.3) / this.PercentRatio.FAT);

                \_ratio.dailyCarbs = Math.round((dailyCalorieIntake \* 0.55) / this.PercentRatio.GARBS);

                return \_ratio;

            }

        }

    }

**Стало:**

public static determineRatioOfPFC(dailyCalorieIntake: number, goalUser: EnumGoalUser): Nutrients {

        switch (goalUser) {

            case EnumGoalUser.LoseWeight: {

                return this.calcRationOfPFC({ protein: 0.3, fats: 0.2, carbs: 0.5 }, dailyCalorieIntake);

            }

            case EnumGoalUser.MaintainWeight: {

                return this.calcRationOfPFC({ protein: 0.3, fats: 0.3, carbs: 0.4 }, dailyCalorieIntake);

            }

            case EnumGoalUser.GainWeight: {

                return this.calcRationOfPFC({ protein: 0.35, fats: 0.3, carbs: 0.55 }, dailyCalorieIntake);

            }

        }

    }

private static calcRationOfPFC(coef: { protein: number; fats: number; carbs: number }, dailyCalorieIntake: number): Nutrients {

        let \_ratio: Nutrients = {

            dailyProtein: 0,

            dailyFat: 0,

            dailyCarbs: 0,

        };

        const { protein, fats, carbs } = coef;

        \_ratio.dailyProtein = Math.round((dailyCalorieIntake \* protein) / this.PercentRatio.PROTEIN);

        \_ratio.dailyFat = Math.round((dailyCalorieIntake \* fats) / this.PercentRatio.FAT);

        \_ratio.dailyCarbs = Math.round((dailyCalorieIntake \* carbs) / this.PercentRatio.GARBS);

        return \_ratio;}

1. Рефакторинг у класі **User:**

**Проблеми**:

* Клас **User** має не дуже добрі назви для розуміння, а також метод **pushCourse** , що дає змогу користувачам підписуватися на курс, потребується у зміні вхідних параметрів ( замість того, щоб зберігати весь курс у інформації користувача, потрібно тільки його id).
* **Rename Method** – назва методу не розкриває суть того, що він робить.
* **Change Parameter** – параметр методу не оптимізован

**Рішення:**

* Необхідно змінити назву методів, а також змінити вхідний параметр на більш підходящий .

**Було:**

export default class User {

    private \_data: dataUser;

    constructor(enterData: dataUser) {

        this.\_data = enterData;

    }

    get about() {

        return this.\_data;

    }

    public pushCourse(course: CourseBase): void {

        this.\_data.courses.push(course.data);

    }

    public removeCourse(courseId: string): void {

        const index = this.\_data.courses.indexOf(courseId);

        console.log(index)

        index > -1 &&  this.\_data.courses.splice(index, 1);

    }

    public hasCurrentCourse(idCourse: string): boolean {

        return this.\_data.courses.some((id) => id === idCourse);

    }

}

**Стало:**

export default class User {

    private \_data: dataUser;

    constructor(enterData: dataUser) {

        this.\_data = enterData;

    }

    get about() {

        return this.\_data;

    }

    public FollowCourse(courseId: string): void {

        this.\_data.courses.push(courseId);

    }

    public UnfollowCourse(courseId: string): void {

        const index = this.\_data.courses.indexOf(courseId);

        console.log(index)

        index > -1 &&  this.\_data.courses.splice(index, 1);

    }

    public hasCurrentCourse(idCourse: string): boolean {

        return this.\_data.courses.some((id) => id === idCourse);

    }

}

1. Рефакторинг у класі **Renderer**. Він займається прийняттям і виконанням функцій і контролерів, які маніпулюють UI у додатку.

**Проблеми**:

* **Long Parameter List (Довгий список параметрів)**  – кількість параметрів методу більше трьох-чотирьох. У нашому випадку конструктор приймає 4 параметри.
* **Extract Method –** У нас є фрагмент коду, який можна згрупувати.

**Рішення:**

* Замість того щоби передавати групу даних, отриманих з іншого об’єкта в якості параметрів, можна передати в метод сам об’єкт, використовуючи передачу всього об’єкта.
* Виділіти цей фрагмент в новий метод (чи функцію) і викликати його замість старого коду.

**Було:**

Клас Renderer:

export default class Renderer {

   private renderHomePage: Function;

   private renderTrainingPage: Function;

   private renderFoodPage: Function;

   private renderProfilePage: Function;

   private controllersUI:Array<Function>;

    constructor(renderHomePage:Function, renderTrainingPage:Function, renderFoodPage:Function,renderProfilePage:Function) {

        this.renderHomePage = renderHomePage;

        this.renderTrainingPage = renderTrainingPage

        this.renderFoodPage = renderFoodPage

        this.renderProfilePage = renderProfilePage

     }

     public renderAllPages(){

        this.renderHomePage();

        this.renderTrainingPage();

        this.renderFoodPage();

        this.renderProfilePage();

     }

     public initControllers(controllersUI:Array<Function>){

      this.controllersUI = controllersUI;

     }

     public startControllers(){

      this.controllersUI.forEach(callback=>{

         callback();

      }) }}

Ініціалізація екземпляру класу:

export const rendererApp = new Renderer(renderHomePage,renderTrainingPage,renderFoodPage,null);

**Стало:**

Клас Renderer:

export default class Renderer {

   private renderPages: renderPagesType;

   private controllersUI: Array<Function>;

   public initRenderPages(renderPages: renderPagesType) {

      this.renderPages = renderPages;

   }

   public renderAllPages() {

      this.renderPages.renderFoodPage();

      this.renderPages.renderHomePage();

      this.renderPages.renderTrainingPage();

      this.renderPages.renderProfilePage();

   }

   public initControllers(controllersUI: Array<Function>) {

      this.controllersUI = controllersUI;

   }

   public startControllers() {

      this.controllersUI.forEach((callback) => {

         callback();

      });

   }

}

Додан тип об’єкту для сторінок:

export type renderPagesType = {

     renderHomePage: Function

     renderTrainingPage: Function

     renderFoodPage: Function

     renderProfilePage: Function

}

Ініціалізація екземпляру класу:

const controllersUI = [addUserController,switchUserControllers,switcherPagesController]

const pages:renderPagesType = {renderFoodPage,renderHomePage,renderTrainingPage,renderProfilePage}

export const rendererApp = new Renderer();

rendererApp.initRenderPages(pages)

rendererApp.initControllers(controllersUI)

rendererApp.startControllers();

1. Рефакторинг у класі **FileSystem:**

**Проблеми**:

* **Rename Method** – назва методу не розкриває суть того, що він робить.

**Рішення:**

* Необхідно змінити назву методів

**Було:**

    public static copyAvatarUser(srcDir:string,destDir:string){

        try {

            fs.copyFileSync(srcDir, destDir)

          } catch (err) {

            console.error(err)

          }

    }

**Стало:**

    public static copyData(srcDir: string, destDir: string) {

        try {

            fs.copyFileSync(srcDir, destDir);

        } catch (err) {

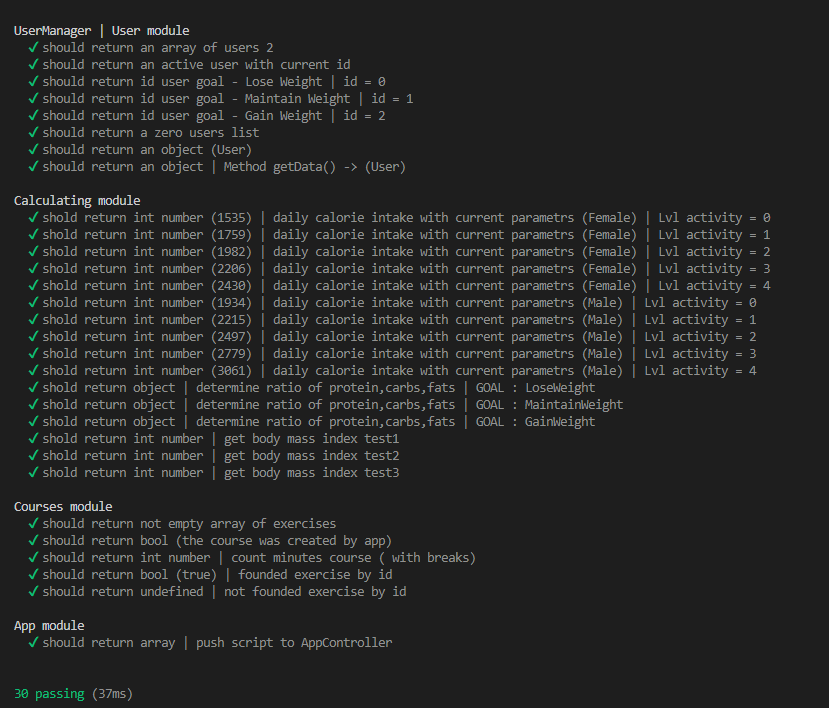
            console.error(err);

        }

    }

Також були видалені непотрібні імпорти та зайві функції, які пов’язані з користувацьким інтерфейсом, переіменовані папки на більш коректні назви, а також завдяки prettier (code formatter) я стилізував код. Після рефакторингу були перевірені Unit-тести.

Всі тести були успішно виконані:



**Висновок:**

Я навчився виконувати реорганізацію програмного коду на підставі шаблонів рефакторінгу. Було внесено список змін у класи додатку. Також були видалені непотрібні імпорти та зайві функції, які пов’язані з користувацьким інтерфейсом, переіменовані папки на більш коректні назви, а також завдяки prettier (code formatter) я стилізував код. Після рефакторингу були перевірені і успішно пройдені Unit-тести.