**ЛАБОРАТОРНА РОБОТА №3**

**ПАТЕРНИ ПРОЄКТУВАВАННЯ**

**Хід роботи:**

Завдання

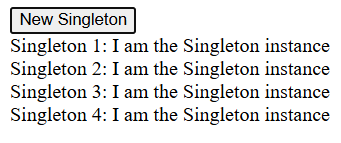
**Для студентів, які мають вищу освіту ступеня “бакалавр” за спеціальністю 121 “Інженерія програмного забезпечення”**

Реалізувати на власний вибір 5 патернів проєктування на будь-якій мові програмування та при захисті пояснити код та суть патернів викладачу.

**Singleton (Одинак)** - породжувальний патерн проектування, який гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього.

**Приклад:**

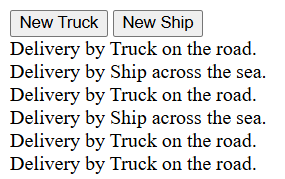
class Singleton {  
 constructor() {  
 if (Singleton.instance) {  
 return Singleton.instance; // Повертаємо вже існуючий екземпляр  
 }  
 Singleton.instance = this; // Зберігаємо новий екземпляр  
 this.data = "I am the Singleton instance"; // Деякі дані в екземплярі  
 }  
  
 // Метод для доступу до даних  
 getData() {  
 return this.data;  
 }  
  
 // Метод для зміни даних  
 setData(newData) {  
 this.data = newData;  
 }  
}  
  
let ***res*** = ***document***.getElementById('result')  
let ***create*** = ***document***.getElementById('createButton')  
let ***counter*** = 1  
// Створювання екземплярів  
  
***create***.onclick = () =>{  
 const singleton = new Singleton();  
 ***res***.innerHTML += "Singleton " + ***counter***.toString() + ": " + singleton.getData() + "<br>"  
 ***counter*** += 1  
}



**Factory Method (Фабричний метод) -** це породжувальний патерн проектування, який визначає загальний інтерфейс для створення об’єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об’єктів.

**Приклад:**

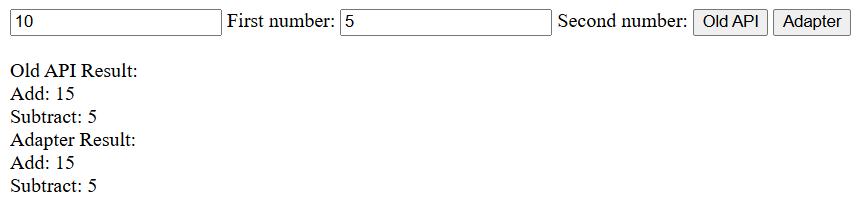
// Базовий клас Транспорту  
class Transport {  
 deliver() {  
 throw new ***Error***("Method 'deliver()' must be implemented.");  
 }  
}  
  
// Клас для транспортування по суші  
class Truck extends Transport {  
 deliver() {  
 return "Delivery by Truck on the road.";  
 }  
}  
  
// Клас для транспортування по морю  
class Ship extends Transport {  
 deliver() {  
 return "Delivery by Ship across the sea.";  
 }  
}  
  
// Фабрика для створення об'єктів транспорту  
class TransportFactory {  
 static *createTransport*(type) {  
 switch (type) {  
 case "truck":  
 return new Truck();  
 case "ship":  
 return new Ship();  
 default:  
 throw new ***Error***("Invalid transport type.");  
 }  
 }  
}  
let ***res*** = ***document***.getElementById('result')  
let ***createTruck*** = ***document***.getElementById('createTruck')  
let ***createShip*** = ***document***.getElementById('createShip')  
// Використання фабрики  
***createTruck***.onclick = () =>{  
 const truck = TransportFactory.*createTransport*("truck");  
 ***res***.innerHTML += truck.deliver() + "<br>"; // "Delivery by Truck on the road."  
}  
***createShip***.onclick = () => {  
 const ship = TransportFactory.*createTransport*("ship");  
 ***res***.innerHTML += ship.deliver() + "<br>"; // "Delivery by Ship across the sea."  
}

****

**Adapter (Адаптер) -** це структурний патерн проектування, що дає змогу об’єктам із несумісними інтерфейсами працювати разом.

**Приклад:**

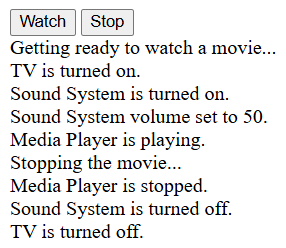
// Старий інтерфейс  
class OldAPI {  
 constructor() {  
 this.operations = function (a, b, operationType) {  
 switch (operationType) {  
 case "add":  
 return a + b;  
 case "subtract":  
 return a - b;  
 default:  
 return ***NaN***;  
 }  
 };  
 }  
}  
  
// Новий інтерфейс  
class NewAPI {  
 add(a, b) {  
 return a + b;  
 }  
  
 subtract(a, b) {  
 return a - b;  
 }  
}  
  
// Адаптер  
class Adapter {  
 constructor() {  
 this.newAPI = new NewAPI();  
 }  
  
 operations(a, b, operationType) {  
 switch (operationType) {  
 case "add":  
 return this.newAPI.add(a, b);  
 case "subtract":  
 return this.newAPI.subtract(a, b);  
 default:  
 return ***NaN***;  
 }  
 }  
}  
let ***res*** = ***document***.getElementById('result')  
let ***num1*** = ***document***.getElementById("num1")  
let ***num2*** = ***document***.getElementById("num2")  
let ***old\_api\_button*** = ***document***.getElementById("Old\_API")  
let ***adapter\_button*** = ***document***.getElementById("Adapter")  
  
// Клієнтський код  
function clientCode(api) {  
 return "Result: <br> Add: " + api.operations(***Number***(***num1***.value), ***Number***(***num2***.value), "add") + // Додає  
 "<br>Subtract: "+ api.operations(***Number***(***num1***.value), ***Number***(***num2***.value), "subtract"); // Віднімає  
}  
  
// Використання старого API  
***old\_api\_button***.onclick = () => {  
 const oldAPI = new OldAPI();  
 ***res***.innerHTML += "<br>Old API " + clientCode(oldAPI);  
}  
  
  
// Використання нового API через адаптер  
***adapter\_button***.onclick = () => {  
 const adaptedAPI = new Adapter();  
 ***res***.innerHTML += "<br>Adapter " + clientCode(adaptedAPI);  
}

****

**Façade (Фасад) -** це структурний патерн проектування, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку

**Приклад:**

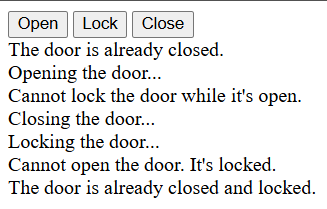
let ***res*** = ***document***.getElementById("result")  
// Підсистеми  
class TV {  
 turnOn() {  
 ***res***.innerHTML += "TV is turned on.<br>";  
 }  
  
 turnOff() {  
 ***res***.innerHTML += "TV is turned off.<br>";  
 }  
}  
  
class SoundSystem {  
 turnOn() {  
 ***res***.innerHTML += "Sound System is turned on.<br>";  
 }  
  
 turnOff() {  
 ***res***.innerHTML += "Sound System is turned off.<br>";  
 }  
  
 setVolume(level) {  
 ***res***.innerHTML += `Sound System volume set to ${level}.<br>`;  
 }  
}  
  
class MediaPlayer {  
 play() {  
 ***res***.innerHTML += "Media Player is playing. <br>";  
 }  
  
 stop() {  
 ***res***.innerHTML += "Media Player is stopped. <br>";  
 }  
}  
  
// Фасад  
class HomeTheaterFacade {  
 constructor() {  
 this.tv = new TV();  
 this.soundSystem = new SoundSystem();  
 this.mediaPlayer = new MediaPlayer();  
 }  
  
 watchMovie() {  
 ***res***.innerHTML += "Getting ready to watch a movie...<br>";  
 this.tv.turnOn();  
 this.soundSystem.turnOn();  
 this.soundSystem.setVolume(50);  
 this.mediaPlayer.play();  
 }  
  
 stopMovie() {  
 ***res***.innerHTML += "Stopping the movie...<br>"  
 this.mediaPlayer.stop();  
 this.soundSystem.turnOff();  
 this.tv.turnOff();  
 }  
}  
  
// Клієнтський код  
const ***homeTheater*** = new HomeTheaterFacade();  
  
// Дивимося фільм  
***document***.getElementById("watch\_movie").onclick = () => ***homeTheater***.watchMovie();  
/\*  
Output:  
Getting ready to watch a movie...  
TV is turned on.  
Sound System is turned on.  
Sound System volume set to 50.  
Media Player is playing.  
\*/  
  
// Зупиняємо фільм  
***document***.getElementById("stop\_movie").onclick = () => ***homeTheater***.stopMovie();  
/\*  
Output:  
Stopping the movie...  
Media Player is stopped.  
Sound System is turned off.  
TV is turned off.  
\*/

****

**State (Стан) -** це поведінковий патерн проектування, що дає змогу об’єктам змінювати поведінку в залежності від їхнього стану. Ззовні створюється враження, ніби змінився клас об’єкта.

**Приклад:**

let ***res*** = ***document***.getElementById("result")  
  
// Стан: Відчинені  
class OpenState {  
 open(door) {  
 ***res***.innerHTML += "The door is already open.<br>";  
 }  
  
 close(door) {  
 ***res***.innerHTML += "Closing the door...<br>";  
 door.setState(new ClosedState());  
 }  
  
 lock(door) {  
 ***res***.innerHTML += "Cannot lock the door while it's open.<br>";  
 }  
}  
  
// Стан: Зачинені  
class ClosedState {  
 open(door) {  
 ***res***.innerHTML += "Opening the door...<br>";  
 door.setState(new OpenState());  
 }  
  
 close(door) {  
 ***res***.innerHTML += "The door is already closed.<br>";  
 }  
  
 lock(door) {  
 ***res***.innerHTML += "Locking the door...<br>";  
 door.setState(new LockedState());  
 }  
}  
  
// Стан: Заблоковані  
class LockedState {  
 open(door) {  
 ***res***.innerHTML += "Cannot open the door. It's locked.<br>";  
 }  
  
 close(door) {  
 ***res***.innerHTML += "The door is already closed and locked.<br>";  
 }  
  
 lock(door) {  
 ***res***.innerHTML += "The door is already locked.<br>";  
 }  
}  
  
// Клас Двері (контекст)  
class Door {  
 constructor() {  
 this.state = new ClosedState(); // Початковий стан дверей  
 }  
  
 setState(state) {  
 this.state = state;  
 }  
  
 open() {  
 this.state.open(this);  
 }  
  
 close() {  
 this.state.close(this);  
 }  
  
 lock() {  
 this.state.lock(this);  
 }  
}  
  
// Використання  
const ***door*** = new Door();  
  
***document***.getElementById('open').onclick = () => ***door***.open();  
***document***.getElementById('lock').onclick = () => ***door***.lock();  
***document***.getElementById('close').onclick = () => ***door***.close();

****