**Лабораторна робота № 5**

**Тема:** Вивчення шаблонів проектування

**Мета:** Навчитись працювати з патернами програмування

**Завдання**

Вивчити та реалізувати шаблони проектування згідно варіанту з використанням TS( https://refactoring.guru/uk/design-patterns/typescript ). Оформити звіт, у якому описати предметну область та проблему, яку вирішуватимете за допомогою шаблона проектування, здійснити опис шаблона проектування, діаграма класів шаблону, реалізація та приклад використання програми(результати роботи).

8. Прототип, Стратегія, Декоратор

**Декоратор** — це структурний патерн проектування, що дає змогу динамічно додавати об’єктам нову функціональність, загортаючи їх у корисні «обгортки».

**Прототип** — це породжувальний патерн проектування, що дає змогу копіювати об’єкти, не вдаючись у подробиці їхньої реалізації.

**Стратегія** — це поведінковий патерн проектування, який визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна заміняти один на інший прямо під час виконання програми.

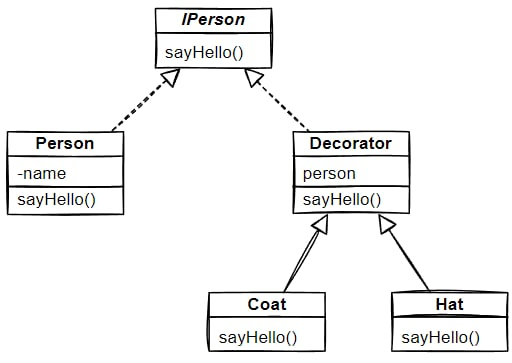
**Хід роботи**

Index.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Лр5 ЛМВ</title>  
</head>  
<body>  
<div class="strategy">  
 <h2>Стратегія</h2>  
 <form id="strategyForm">  
 <label for="distance">Відстань</label>  
 <input id="distance" type="number" value="1" min="1"/>  
 <label for="way">Спосіб</label>  
 <select name="way" id="way">  
 <option value="taxi">Таксі</option>  
 <option value="publicTransport">Публічний транспорт</option>  
 <option value="walking">Пішки</option>  
 </select>  
 <button type="submit">Готово</button>  
 </form>  
</div>  
<div class="prototype">  
 <h2>Прототип</h2>  
 <form id="prototypeForm">  
 <label for="lastname">Прізвище</label>  
 <input id="lastname" type="text" required>  
 <label for="firstname">Ім'я</label>  
 <input id="firstname" type="text" required>  
 <label for="group">Група</label>  
 <input id="group" type="text" required>  
 <button type="submit">Створити</button>  
 </form>  
</div>  
<div class="decorator">  
 <h2>Декоратор</h2>  
 <img src="images/person.png" alt="" id="person">  
 <button id="addCoat">Одягти куртку</button>  
 <button id="addHat">Одягти шапку</button>  
</div>  
<script src="strategy.js"></script>  
<script src="prototype.js"></script>  
<script src="decorator.js"></script>  
</body>  
</html>

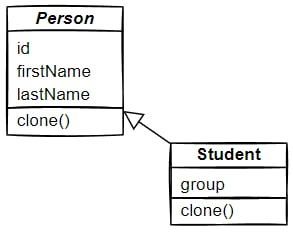
decorator.ts

interface PersonI {  
 sayHello(): string;  
}  
  
class Person implements PersonI {  
 public name: string;  
  
 constructor(name) {  
 this.name = name;  
 }  
  
 public sayHello(): string {  
 return 'It is too cold to say Hello';  
 }  
}  
  
class Decorator implements PersonI {  
 protected person: Person;  
  
 constructor(person: Person) {  
 this.person = person;  
 }  
  
 public sayHello(): string {  
 return this.person.sayHello();  
 }  
}  
  
class Coat extends Decorator {  
  
 public sayHello(): string {  
 return `Hello, thank you for coat`;  
 }  
}  
  
class Hat extends Decorator {  
 public sayHello(): string {  
 return `${super.sayHello()}. I have great hat`;  
 }  
}  
  
let ***person*** = new Person('Мирослав');  
const ***addCoat*** = ***document***.getElementById('addCoat');  
const ***addHat*** = ***document***.getElementById('addHat');  
***addCoat***.addEventListener('click', () => {  
 // @ts-ignore  
 ***person*** = new Coat(***person***);  
 ***console***.log(***person***);  
 changeImage(***person***);  
 // @ts-ignore  
 ***addCoat***.disabled = true;  
})  
  
***addHat***.addEventListener('click', () => {  
 // @ts-ignore  
 ***person*** = new Hat(***person***);  
 ***console***.log(***person***);  
 changeImage(***person***);  
 // @ts-ignore  
 ***addHat***.disabled = true;  
})  
  
const ***img*** = ***document***.getElementById('person');  
***img***.addEventListener('click', () => {  
 alert(***person***.sayHello());  
});  
  
function changeImage(person) {  
 if (person instanceof Person) {  
 // @ts-ignore  
 ***img***.src = 'images/person.png'  
 } else {  
 // @ts-ignore  
 if (person instanceof Coat && person.person instanceof Person) {  
 // @ts-ignore  
 ***img***.src = 'images/personWithCoat.png';  
 return;  
 }  
 // @ts-ignore  
 if (person instanceof Hat && person.person instanceof Person) {  
 // @ts-ignore  
 ***img***.src = 'images/personWithHat.png';  
 return;  
 }  
 // @ts-ignore  
 ***img***.src = 'images/personWithHatAndCoat.png';  
 }  
}



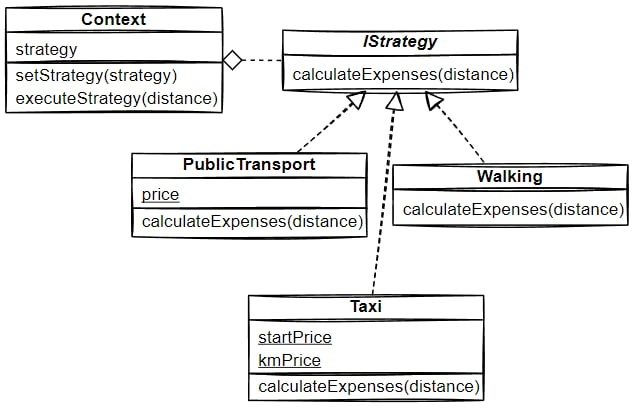
prototype.ts

const hashCode = function (str) {  
 let hash = 0, i, chr;  
 if (str.length === 0) return hash;  
 for (i = 0; i < str.length; i++) {  
 chr = str.charCodeAt(i);  
 hash = ((hash << 5) - hash) + chr;  
 hash |= 0; // Convert to 32bit integer  
 }  
 return hash;  
};  
  
abstract class Person {  
 private readonly id: number;  
 public firstName: string;  
 public lastName: string;  
  
 protected constructor(source) {  
 this.id = hashCode(`${source.lastName} ${source.firstName}`);  
 this.firstName = source.firstName;  
 this.lastName = source.lastName;  
 }  
  
 abstract clone(): Person;  
}  
  
class Student extends Person {  
 public group: string;  
  
 constructor(source: Student) {  
 super(source);  
 this.group = source.group;  
 }  
  
 clone(): Student {  
 return new Student(this)  
 }  
}  
  
const ***prototypeForm*** = ***document***.getElementById('prototypeForm');  
const ***lastNameInput*** = ***document***.getElementById('lastname');  
const ***firstNameInput*** = ***document***.getElementById('firstname');  
const ***groupInput*** = ***document***.getElementById('group');  
***prototypeForm***.addEventListener('submit', (e) => {  
 e.preventDefault();  
 // @ts-ignore  
 const student = new Student({firstName: ***firstNameInput***.value, lastName: ***lastNameInput***.value, group: ***groupInput***.value});  
 ***console***.log(student.clone());  
 console.log({...student});  
})

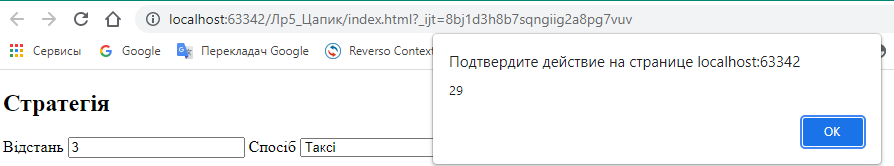


strategy.ts

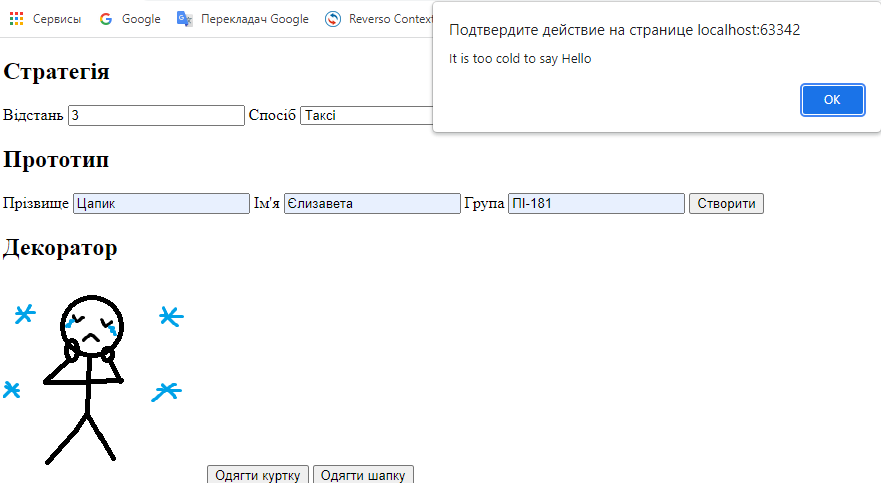
interface Strategy {  
 calculateExpenses(distance);  
}  
  
class PublicTransport implements Strategy {  
 private static *price* = 8;  
  
 calculateExpenses(distance) {  
 return distance < 10 ? PublicTransport.*price* : distance / 10 \* PublicTransport.*price*;  
 }  
}  
  
class Walking implements Strategy {  
 calculateExpenses(distance) {  
 return 0;  
 }  
}  
  
class Taxi implements Strategy {  
 private static *startPrice* = 20;  
 private static *kmPrice* = 3;  
  
 calculateExpenses(distance) {  
 return Taxi.*startPrice* + Taxi.*kmPrice* \* distance;  
 }  
}  
  
class Context {  
 private strategy: Strategy;  
  
 setStrategy(strategy: Strategy) {  
 this.strategy = strategy  
 }  
  
 executeStrategy(distance) {  
 return this.strategy.calculateExpenses(distance)  
 }  
}  
  
const ***way*** = {  
 'taxi': new Taxi(),  
 'publicTransport': new PublicTransport(),  
 'walking': new Walking()  
}  
  
const ***strategyForm*** = ***document***.getElementById('strategyForm');  
const ***distanceInput*** = ***document***.getElementById('distance');  
const ***waySelect*** = ***document***.getElementById('way');  
***strategyForm***.addEventListener('submit', (e) => {  
 e.preventDefault();  
 const context = new Context();  
 // @ts-ignore  
 context.setStrategy(***way***[***waySelect***.value]);  
 // @ts-ignore  
 const result = context.executeStrategy(***distanceInput***.value);  
 ***alert***(result)  
})

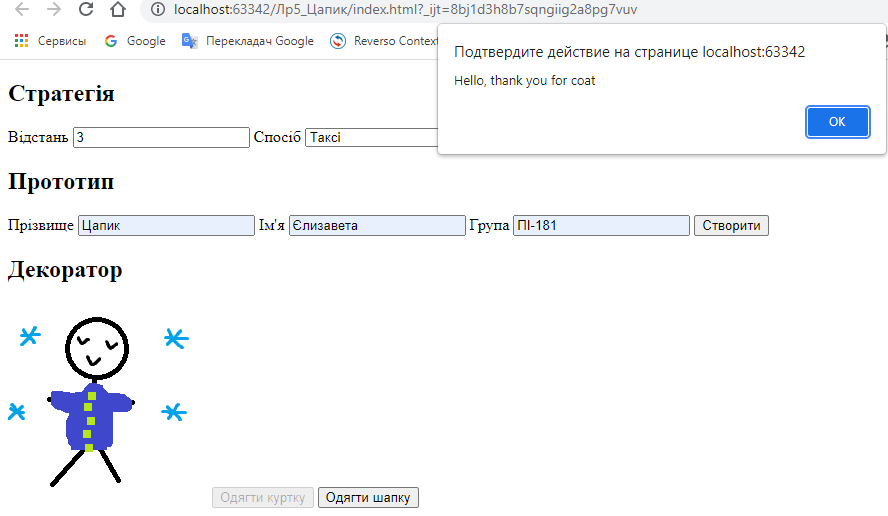


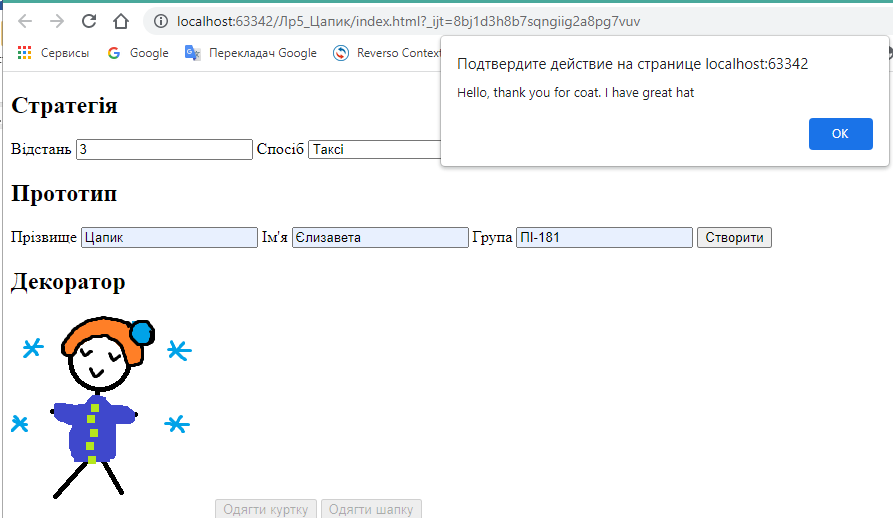
Результат











**Висновок:** Навчилась працювати з патернами програмування