Ministerul Educației al Republicii Moldova

Universitatea Tehnica a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

Departamentul Inginerie Software si Automatica

**Raport**

Laborator nr. 4

Disciplina: Tehnici si mecanisme de proiectare software.

Tema: „Behavioral Design Patterns.”

A efectuat: st.gr.TI–202

Rusu Felicia

A verificat: asist.univ.

Gaidau Mihai

Chişinău 2023

**Obiective:**

1. Studiați și înțelegeți modelele de design comportamental.

2. Ca o continuare a lucrărilor anterioare de laborator, gândiți-vă la ce comunicare între entitățile software ar putea fi implicată în sistemul dumneavoastră.

3. Creați un proiect nou sau adăugați câteva funcționalități suplimentare folosind modele de design comportamental.

Modelele de design comportamental sunt o categorie de modele de design care se concentrează pe interacțiunea și comunicarea dintre obiecte și clase. Ele oferă o modalitate de a organiza comportamentul obiectelor într-un mod flexibil și reutilizabil, separând în același timp responsabilitățile obiectelor de detaliile specifice de implementare. Tiparele de design comportamental abordează probleme comune întâlnite în comportamentul obiectelor, cum ar fi modul de definire a interacțiunilor dintre obiecte, modul de control al fluxului de mesaje între obiecte sau modul de definire a algoritmilor și politicilor care pot fi reutilizate în diferite obiecte și clase.

Câteva exemple din această categorie de modele de design sunt:

* Chain of Responsibility - obiectele de comandă sunt gestionate sau transmise altor obiecte de obiecte de procesare care conțin logică.
* Command - obiectele de comandă încapsulează o acțiune și parametrii acesteia.
* Interpreter - implementați un limbaj de calculator specializat pentru a rezolva rapid un anumit set de probleme.
* Iterator - iteratorii sunt utilizați pentru a accesa secvențial elementele unui obiect agregat fără a expune reprezentarea de bază.
* Mediator - oferă o interfață unificată unui set de interfețe dintr-un subsistem.
* Observer - Obiectele se înregistrează pentru a observa un eveniment care poate fi ridicat de un alt obiect.
* Strategy - algoritmii pot fi selectați din mers, folosind compoziția.

In aceasta lucrare am folosit doua modele de design de tip comportamental precum Observer si Strategy.

Model de observator: clasa Student acționează ca un observator și se înregistrează/se anulează înregistrarea la un obiect Examen pentru a primi notificări despre modificările din examen.

Model de strategie: interfața IExamEvaluationStrategy este utilizată pentru a defini o strategie abstractă pentru evaluarea examenelor și sunt create două clase concrete ExamEvaluationStrategy1 și ExamEvaluationStrategy2 care implementează această interfață pentru a oferi diferiți algoritmi de evaluare. Acest lucru permite ca algoritmul de evaluare să fie comutat în timpul execuției fără a afecta clienții care folosesc algoritmul.

Codul furnizat este scris în C# și constă din patru clase: Program, Student, IExamEvaluationStrategy, ExamEvaluationStrategy1 și ExamEvaluationStrategy2.

Clasa Program.cs:

using System;

using System.Collections.Generic;

using lab4\_Behavioral\_DP;

class Program

{

static void Main(string[] args)

{

// Crearea unui examen

Exam exam = new Exam("C# Programming Exam", DateTime.Parse("2023-06-01 10:00"), "Aula Magna");

// Crearea studenților

Student student1 = new Student("John", "Doe", "jd123");

Student student2 = new Student("Jane", "Doe", "jd456");

// Înregistrarea studenților la examen

exam.Register(student1);

exam.Register(student2);

// Notificarea studenților despre schimbările examenului

exam.ChangeDateTime(DateTime.Parse("2023-06-02 14:00"));

exam.ChangeLocation("Sala 101");

// Evaluarea examenului folosind două strategii de evaluare diferite

List<int> grades1 = new List<int> { 8, 9, 7, 8, 10, 9 };

IExamEvaluationStrategy evaluationStrategy1 = new ExamEvaluationStrategy1();

double result1 = evaluationStrategy1.Evaluate(grades1);

Console.WriteLine($"Media examenului (strategia 1): {result1}");

List<int> grades2 = new List<int> { 7, 6, 9, 8, 5, 7 };

IExamEvaluationStrategy evaluationStrategy2 = new ExamEvaluationStrategy2();

double result2 = evaluationStrategy2.Evaluate(grades2);

Console.WriteLine($"Media examenului (strategia 2): {result2}");

}

}

Clasa Student.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab4\_Behavioral\_DP

{

public class Student

{

public Student(string v1, string v2, string v3)

{

V1 = v1;

V2 = v2;

V3 = v3;

}

public string Name { get; set; }

public string V1 { get; }

public string V2 { get; }

public string V3 { get; }

public void Subscribe(Exam exam)

{

exam.Subscribe(this);

}

public void Unsubscribe(Exam exam)

{

exam.Unsubscribe(this);

}

public void ReceiveNotification(string message)

{

Console.WriteLine($"Notificare pentru studentul {Name}: {message}");

}

}

}

Clasa IExamEvaluationStrategy.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab4\_Behavioral\_DP

{

public interface IExamEvaluationStrategy

{

double Evaluate(List<double> grades);

double Evaluate(List<int> grades1);

}

}

Clasa ExamEvaluationStrategy1.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab4\_Behavioral\_DP

{

public class ExamEvaluationStrategy1 : IExamEvaluationStrategy

{

public double Evaluate(List<double> grades)

{

double sum = 0;

foreach (var grade in grades)

{

sum += grade;

}

return sum / grades.Count;

}

public double Evaluate(List<int> grades)

{

double result = 0;

// Implementarea logicii pentru evaluarea examenului

foreach (int grade in grades)

{

result += grade;

}

result /= grades.Count;

result = Math.Round(result, 2);

if (result < 5)

{

Console.WriteLine("Studentul a picat examenul.");

}

else if (result >= 5 && result < 7)

{

Console.WriteLine("Studentul a luat un 5 sau un 6.");

}

else if (result >= 7 && result < 8.5)

{

Console.WriteLine("Studentul a luat un 7 sau un 8.");

}

else if (result >= 8.5 && result < 9.5)

{

Console.WriteLine("Studentul a luat un 8.5 sau un 9.");

}

else

{

Console.WriteLine("Studentul a luat un 10.");

}

return result;

}

}

}

Clasa ExamEvaluationStrategy2.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab4\_Behavioral\_DP

{

public class ExamEvaluationStrategy2 : IExamEvaluationStrategy

{

public double Evaluate(List<double> grades)

{

double minGrade = grades.Min();

double maxGrade = grades.Max();

double sum = 0;

foreach (var grade in grades)

{

sum += grade;

}

return (sum - minGrade - maxGrade) / (grades.Count - 2);

}

public double Evaluate(List<int> grades)

{

double result = 0;

// Implementarea logicii pentru evaluarea examenului

foreach (int grade in grades)

{

result += grade;

}

result /= grades.Count;

result = Math.Round(result, 2);

if (result < 6)

{

Console.WriteLine("Studentul a picat examenul.");

}

else if (result >= 6 && result < 8)

{

Console.WriteLine("Studentul a luat un 6 sau un 7.");

}

else if (result >= 8 && result < 9.5)

{

Console.WriteLine("Studentul a luat un 8 sau un 9.");

}

else

{

Console.WriteLine("Studentul a luat un 10.");

}

return result;

}

}

}

Clasa Exam.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab4\_Behavioral\_DP

{

public class Exam

{

private readonly List<Student> students = new List<Student>();

public Exam(string v1, DateTime dateTime, string v2)

{

V1 = v1;

DateTime = dateTime;

V2 = v2;

}

public string Subject { get; set; }

public DateTime Date { get; set; }

public string Location { get; set; }

public string V1 { get; }

public DateTime DateTime { get; }

public string V2 { get; }

public void Subscribe(Student student)

{

students.Add(student);

}

public void Unsubscribe(Student student)

{

students.Remove(student);

}

public void NotifyStudents(string message)

{

foreach (var student in students)

{

student.ReceiveNotification($"{Subject} la data de {Date} la locatia {Location}: {message}");

}

}

internal void ChangeDateTime(DateTime dateTime)

{

}

internal void ChangeLocation(string v)

{

}

internal void Register(Student student1)

{

}

}

}

Clasa Program conține metoda Main care creează o instanță a unui examen, înregistrează doi studenți la examen, schimbă data și locația examenului și evaluează examenul folosind două strategii diferite.

Clasa Student reprezintă un student și conține numele, prenumele și ID-ul acestuia. Clasa conține, de asemenea, metode de abonare și dezabonare la un examen și de a primi notificări despre modificările examenului.

Interfața IExamEvaluationStrategy definește contractul pentru strategiile de evaluare a examenelor. Conține două metode care acceptă o listă de note și returnează scorul mediu al examenului.

Clasele ExamEvaluationStrategy1 și ExamEvaluationStrategy2 implementează interfața IExamEvaluationStrategy și conțin logica pentru evaluarea unui examen folosind două strategii diferite. ExamEvaluationStrategy1 calculează scorul mediu și rotunjește rezultatul la două zecimale. Apoi tipărește un mesaj bazat pe scorul calculat. ExamEvaluationStrategy2 calculează scorul mediu excluzând notele cele mai mari și cele mai mici și rotunjește rezultatul la două zecimale. Apoi tipărește un mesaj bazat pe scorul calculat.

Clasa principală este clasa Program, care creează o instanță a clasei Exam, înregistrează două obiecte Student și demonstrează utilizarea a două implementări diferite IExamEvaluationStrategy pentru a evalua rezultatele examenului.

Clasa de examen gestionează înregistrarea și notificarea studenților, precum și posibilitatea de a schimba data și locația examenului. Implementează interfața ISubject, care definește metodele de abonare și dezabonare a observatorilor, precum și de notificare a acestora cu privire la modificări.

Clasa Student reprezintă un elev, cu nume și ID. Implementează interfața IObserver, care definește metoda de primire a notificărilor de la un subiect.

Interfața IExamEvaluationStrategy definește metodele de evaluare a unui examen, cu două supraîncărcări pentru a suporta diferite tipuri de intrare (listă de duble sau listă de numere întregi).

Clasele ExamEvaluationStrategy1 și ExamEvaluationStrategy2 implementează interfața IExamEvaluationStrategy cu o logică de evaluare diferită pentru rezultatele examenului.

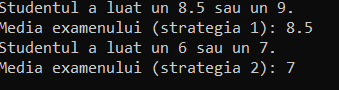


Figura1 – Rezultatul executiei a acestui proiect

**Concluzie:**

În concluzie, utilizarea modelelor de proiectare în dezvoltarea de software oferă mai multe beneficii, inclusiv vizibilitatea, mentenabilitatea și scalabilitatea îmbunătățite ale codului. Urmând cele mai bune practici dovedite, dezvoltatorii pot proiecta soluții software care sunt mai ușor de înțeles, modificat și extins în timp. Deși există multe modele de proiectare disponibile, alegerea modelului potrivit pentru o anumită problemă poate fi o provocare și necesită o analiză atentă a diferiților factori, cum ar fi natura problemei, arhitectura sistemului și abilitățile și experiența echipei de dezvoltare. În cele din urmă, utilizarea eficientă a modelelor de design poate ajuta dezvoltatorii să creeze sisteme software robuste și fiabile, care să răspundă nevoilor utilizatorilor și ale părților interesate.