Universitatea Tehnică a Moldovei

Facultatea Calculatoare Informatică și Microelectronică

Departamentul Inginerie Software și Automatică

RAPORT

Proiect de AN la TMPS



A efectuat: Ciumac Nichita st. gr. TI-204

A verificat: Gaidau Mihai

Chişinău – 2023

**Cuprins**

**Introducere............................................................................................................................................3**

**Caracteristicele.......................................................................................................................................4**

**Design Pattern................................................................................................................................5**

**Modele de design.....................................................................................................................................6**

**Observer Design Pattern /Command Design Pattern…………………………………………………………………..7**

**Strategy Design Pattern/ Mediator Design Pattern……………………………………………………………………...8**

**Diagrame UML.......................................................................................................................................9**

**Observer Design Pattern……………………………………………………………………….10**

**Command Design Pattern.....................................................................................................................11**

**Mediator Design Pattern………………………………………………………………………………12**

**Concluzie.................................................................................................................................................14**

**Bibliografie..............................................................................................................................................15**

# 

# **Introducere**

Aplicația "ConversieUnități" oferă utilizatorilor posibilitatea de a efectua conversii rapide și precise între diferite unități de măsură. Indiferent dacă este vorba de transformarea metrilor și centimetrilor în picioare și inchi, de convertirea picioarelor și inchilor în metri și centimetri, de conversia kilometrilor în mile sau de schimbarea milelor în kilometri, această aplicație ușor de utilizat vă permite să efectuați aceste conversii într-un mod simplu și eficient.

Aplicația implementează patru design patterns care contribuie la organizarea și eficiența codului. Observer Design Pattern este utilizat pentru a actualiza automat rezultatele conversiilor și a le afișa pe ecran. Pattern-ul Command este folosit pentru a reprezenta fiecare conversie ca o comandă independentă și pentru a permite extensibilitatea ulterioară prin adăugarea altor tipuri de conversii. De asemenea, Mediator Design Pattern este utilizat pentru a gestiona execuția comenzilor și pentru a asigura o interacțiune coerentă între utilizator și sistem. În plus, codebase-ul este organizat într-o structură modulară, utilizând conceptul de clase și obiecte pentru a îmbunătăți reutilizarea și întreținerea codului.

Aplicația "ConversieUnități" reprezintă o soluție practică și eficientă pentru conversia rapidă și precisă a unităților de măsură. Prin intermediul acestei aplicații, utilizatorii vor putea să efectueze conversii în mod convenabil, eliminând necesitatea de a face calcule complexe manual.

Scopul aplicației "ConversieUnități" este de a oferi utilizatorilor o modalitate simplă și eficientă de a efectua conversii între diferite unități de măsură. Aceasta are ca scop facilitarea transformării rapide și precise a valorilor dintr-o unitate de măsură în alta, eliminând necesitatea de a efectua calcule complexe manual.

**Caracteristicile**

Caracteristicile principale ale acestor aplicații sunt:

1 . Conversii multiple: Aplicația permite utilizatorilor să efectueze conversii între diverse unități de măsură, inclusiv metri și centimetri în picioare și inchi, picioare și inchi în metri și centimetri, kilometri în mile și mile în kilometri. Astfel, utilizatorii pot converti valorile în funcție de nevoile lor specifice.

2. Interfață ușor de utilizat: Aplicația oferă o interfață simplă și simplă, care permite utilizatorilor să introducă valorile de conversie și să obțină intuitivă în mod rapid și fără efort. Utilizatorii pot selecta opțiunea de conversie dorită și pot introduce valorile în unitățile respective.

3. Rezultate precise: Conversiile efectuate de aplicare sunt bazate pe formule matematice precise și sunt afișate cu o precizie adecvată. Astfel, utilizatorii pot avea încredere că obținute sunt corecte și acurate.

4. Actualizări în timp real: Prin utilizarea design pattern-ului Observer, aplicația poate actualiza automate conversii și le poate afișa pe ecran în timp real. Astfel, utilizatorii pot vedea instantaneu rezultatul conversiei fără a fi nevoie să reîncarce sau să reintroducă datele.

5. Extensibilitate: Aplicația utilizat design pattern-ul Command, permițând extinderea ușoară a funcționalității prin adăugarea altor tipuri de conversii în viitor. Astfel, aplicația poate fi adaptată pentru a suporta noi unități de măsură sau alte operații specifice.

Prin urmare, aplicația "ConversieUnități" oferă utilizatorilor un instrument practic și eficient pentru a efectua conversii între unități de măsură diferite, asigurând rezultate precise și ușurând procesul de transformare a valorilor.

**Design Pattern**

Un design pattern (sau model de proiectare) este o soluție generală și reutilizabilă pentru o problemă comună care apare în dezvoltarea software-ului. Acestea sunt considerate soluții practice, testate și validate care ajută la rezolvarea unor probleme comune de proiectare și implementare într-un mod eficient și robust.

Design patterns-ul furnizează un set de reguli, principii și abstractii care permit dezvoltatorilor să creeze structura flexibile, extensibile și ușor de întreținut. Ele descriu relațiile între obiecte, clase sau module și oferă un cadru conceptual pentru a aborda anumite probleme specifice.

Există multe tipuri de design patterns, împărțite în diferite categorii, cum ar fi crearea, structurale sau comportamentale. Fiecare tip de design pattern are un scop specific și abordează o problemă particulară. De exemplu, un design pattern de tipul "Singleton" este utilizat pentru a asigura că o clasă are o singură instanță în cadrul unei aplicații, în timp ce un design pattern de tipul "Observer" permite notificarea automată a unor obiecte despre modificările apărute într-un loc. un alt obiect.

Beneficiile utilizării design patterns-urilor includ:

1. Reutilizarea soluțiilor testate și valide
2. Îmbunătățirea flexibilității și extensibilității codului
3. Îmbunătățirea înțelegerii și comunicării între dezvoltatori
4. Promovarea principiilor de proiectare solidă și modularitate

Design patterns-urile de un instrument pentru dezvoltatorii de software și pentru a ajuta la crearea unor aplicații robuste, ușor de întreținut și scalabile.

**Modele de design**

În acest program au fost folosite următoarele modele de design:

1. Observer Design Pattern (Modelul Observatorului) - utiliza pentru notificarea automată a observatorilor atunci când se schimbă starea subiectului.
2. Command Design Pattern (Modelul Comenzii) - utilizați pentru încapsularea comenzilor de conversie în obiecte independente și execuția lor flexibilă.
3. Strategy Design Pattern (Modelul Strategiei) - utilizați pentru definirea și utilizarea flexibilă a diferitelor strategii de conversie.
4. Mediator Design Pattern (Modelul Mediatorului) - utilizați pentru gestionarea interacțiunilor dintre utilizator și execuția comenzilor de conversie.

Aceste design patterns sunt folosite în program pentru a asigura o structură clară, modularitate și flexibilitate în implementarea convertitorilor și interacțiunilor cu utilizatorul. Ele aduc beneficii precum decuplarea componentelor, reutilizarea codului și ușurința în extinderea funcționalității.

**Observer Design Pattern /Command Design Pattern**

**Observer Design Pattern (Modelul Observatorului):**

Acest pattern permite implementarea unui sistem de notificare în care un subiect notifică automat mai mulți observatori despre schimbările stării sale.

Observatorii sunt înregistrați la subiect și primesc notificări atunci când se produc schimbări relevante în subiect.

Acest pattern permite o cuplare slabă între subiect și observatori, permițând astfel flexibilitate în adăugarea sau eliminarea observatorilor fără a afecta subiectul.

În codul dat, Observer Design Pattern este utilizat pentru ca observatorul ConsoleOutput să primească și să afișeze rezultatele de conversie furnizate de subiectul ConversieSubject.

**Command Design Pattern (Modelul Comenzii):**

Acest pattern separă solicitarea unei acțiuni de execuția acesteia, utilizând o clasă de comandă intermediară.

O comandă încapsulează o anumită acțiune și include informațiile necesare pentru a o realiza, precum și obiectul asupra căruia se aplică acțiunea și parametrii necesari.

Prin utilizarea acestui pattern, se poate obține o mai mare flexibilitate în gestionarea comenzilor, precum compunerea și decuplarea acestora în mod dinamic.

În codul dat, Command Design Pattern este folosit pentru a încapsula conversiile specifice în obiecte de comandă ConversieCommand, care sunt apoi executate de mediatorul Mediator în momentul potrivit.

**Strategy Design Pattern/ Mediator Design Pattern**

**Strategy Design Pattern (Modelul Strategiei):**

Acest pattern permite definirea mai multor algoritmi diferiți și încapsularea fiecărui algoritm într-o clasă separată.

Algoritmii pot fi utilizați interșanjabili în funcție de nevoile și contextul aplicației, fără a afecta codul client.

Acest pattern promovează flexibilitatea și extensibilitatea prin posibilitatea schimbării algoritmului în timpul execuției.

În codul dat, Strategy Design Pattern este utilizat pentru a defini strategii de conversie diferite, cum ar fi conversia metri în centimetri, picioare în inchi, mile în kilometri și kilometri în mile. Fiecare strategie este reprezentată de o clasă separată, cum ar fi MetriCentimetriStrategy, PicioareInchiStrategy, MileKilometriStrategy și KilometriMileStrategy.

**Mediator Design Pattern (Modelul Mediatorului):**

Acest pattern facilitează comunicarea și interacțiunea între obiecte prin intermediul unui obiect mediator central.

Mediatorul acționează ca o componentă intermediară care coordonează și controlează interacțiunea dintre obiectele participante, reducând astfel dependența directă între ele.

Utilizarea unui mediator permite obținerea unei structuri mai flexibile și mai modularizate, în care obiectele comunică prin intermediul mediatorului, fără a fi nevoie să se cunoască reciproc.

În codul dat, Mediator Design Pattern este utilizat pentru a realiza comunicarea între utilizator, comenzi și subiectul de conversie. Mediatorul Mediator primește comanda de la utilizator și o transmite către subiectul de conversie, care apoi notifică observatorul ConsoleOutput cu rezultatul conversiei.

# **Diagrame UML**

O diagramă UML (Unified Modeling Language) este o reprezentare grafică a diferitelor aspecte ale unui sistem software sau al unui proces de dezvoltare software. UML este un limbaj standardizat utilizat în ingineria software pentru a modela, proiecta și documenta sistemele software.

Diagrama UML utilizează o serie de simboluri grafice pentru a reprezenta entități, relații și interacțiuni între acestea. Aceste diagrame facilitează comunicarea și înțelegerea între membrii echipei de dezvoltare software, precum și între echipele interdisciplinare.

Există mai multe tipuri de diagrame UML, fiecare având un scop specific. Acestea includ diagrame de clasă, diagrame de obiecte, diagrame de secvențe, diagrame de activități, diagrame de stări, diagrame de componentă, diagrame de pachet și altele. Fiecare diagramă UML oferă o perspectivă diferită asupra sistemului și evidențiază diferite aspecte și relații.

Există mai multe tipuri de diagrame UML, fiecare cu scopul și obiectivul său specific. Iată câteva tipuri comune de diagrame UML:

1. Diagrama de clasă: reprezintă structura statică a unui sistem, inclusiv clasele, atributele, metodele și relațiile acestora.
2. Diagrama de caz de utilizare: ilustrează interacțiunile dintre actori (utilizatori sau sisteme externe) și sistem, concentrându-se pe funcționalitatea sistemului și pe obiectivele utilizatorului.
3. Diagrama secvenței: arată interacțiunile dintre obiecte de-a lungul timpului, captând ordinea mesajelor schimbate între ele.
4. Diagrama activității: Prezintă fluxul activităților sau proceselor dintr-un sistem, prezentând secvența acțiunilor și deciziilor.
5. Diagrama mașinii de stări: reprezintă diferitele stări și tranziții ale unui obiect sau sistem, captând comportamentul acestuia ca răspuns la evenimente.
6. Diagrama componentelor: Vizualizează componentele fizice sau logice ale unui sistem și dependențele acestora.
7. Diagrama de implementare: arată componentele hardware și software dintr-un sistem și aranjarea lor fizică într-o rețea sau într-un mediu de calcul.

Aceste diagrame, împreună cu altele precum Diagramele pachetelor, Diagramele de comunicare și Diagramele de colaborare, oferă o vedere cuprinzătoare a arhitecturii, designului și interacțiunilor unui sistem.

Prin utilizarea diagramelor UML, profesioniștii în software își pot comunica și împărtăși ideile în mod eficient, pot analiza și valida proiectele de sistem și pot facilita colaborarea între membrii echipei implicați în dezvoltarea și întreținerea sistemului.

**Observer Design Pattern:**

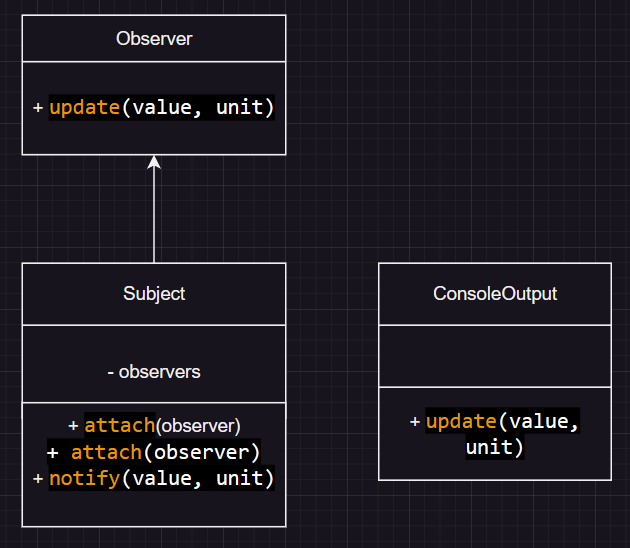


Figura 1. Această diagramă reprezintă design pattern-ul Observer utilizat în program.

Observer este o interfață care definește metoda update(value, unit) ce trebuie implementată de toți observatorii.

Subject este o interfață care definește metodele attach(observer), detach(observer) și notify(value, unit) pentru gestionarea observatorilor.

ConsoleOutput este o clasă concretă ce implementează interfața Observer și reprezintă un observator specific care afișează rezultatele pe consolă.

Relația dintre Subject și Observer este una de tipul "one-to-many". Subiectul (Subject) poate avea mai mulți observatori (Observer) atașați, iar atunci când are loc o actualizare, subiectul notifică toți observatorii prin apelul metodei update(value, unit).

Astfel, prin utilizarea acestui design pattern, se permite obținerea de notificări și actualizări în timp real atunci când apar modificări în subiect. Observatorii pot fi adăugați și înlăturați dinamic și pot reacționa în mod individual la evenimente. În cazul programului dat, ConsoleOutput este observatorul care primește actualizările și le afișează pe consolă.

**Command Design Pattern:**

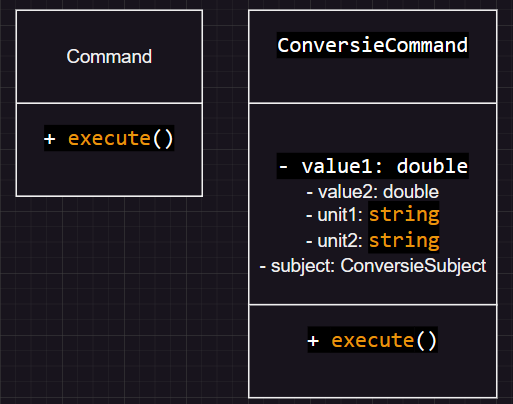


Figura 1.2.Această diagramă reprezintă design pattern-ul Command utilizat în program.

Command este o interfață care definește metoda execute() ce trebuie implementată de toate comenzile.

ConversieCommand este o clasă concretă ce implementează interfața Command și reprezintă o comandă specifică de conversie. Această clasă conține variabilele value1, value2, unit1, unit2 care reprezintă valorile și unitățile de conversie și o referință către obiectul ConversieSubject.

Metoda execute() din clasa ConversieCommand calculează rezultatul conversiei și notifică obiectul ConversieSubject cu ajutorul metodei notify(value, unit).

Prin utilizarea acestui design pattern, se permite encapsularea comenzilor intr-un obiect separat, care poate fi executat independent de solicitant. Astfel, se realizează o separare între logica de apelare a comenzilor și implementarea acestora.

În contextul programului dat, ConversieCommand reprezintă comanda de conversie și este executată de obiectul Mediator. Comanda primește valorile și unitățile de conversie, efectuează calculele necesare și notifică obiectul ConversieSubject cu rezultatul obținut.

**Strategy Design Pattern**

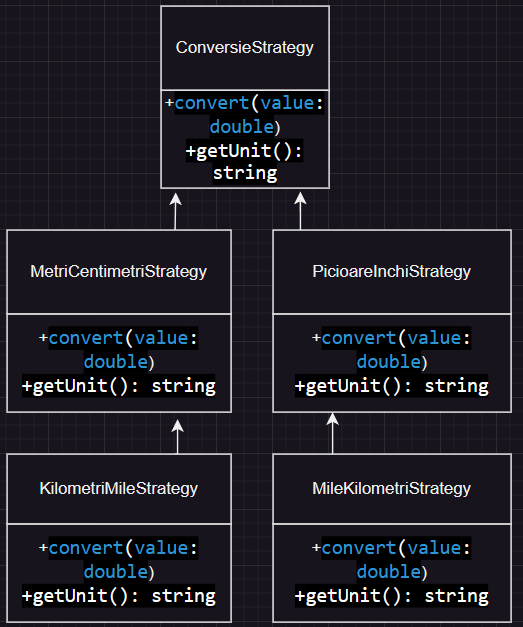


Figura 1.3. Această diagramă reprezintă design pattern-ul Strategy folosit în programul dat.

ConversieStrategy este o interfață care definește metodele convert(value: double) și getUnit(): string, care trebuie implementate de toate strategiile de conversie.

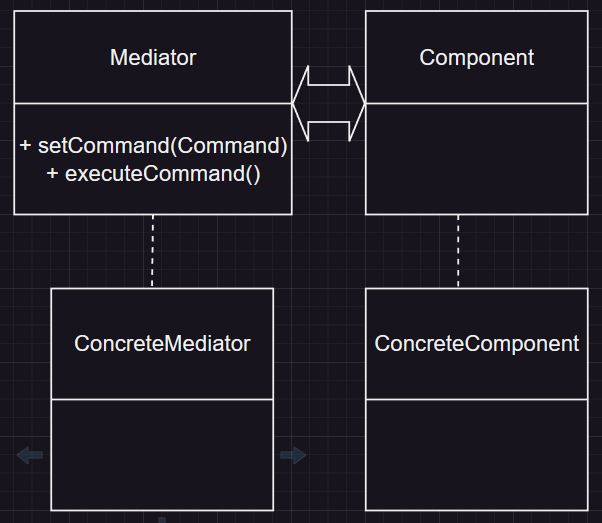
Există patru clase concrete care implementează interfața ConversieStrategy: MetriCentimetriStrategy, PicioareInchiStrategy, KilometriMileStrategy și MileKilometriStrategy. Aceste clase reprezintă diferite strategii de conversie și conțin implementări specifice ale metodelor convert(value) și getUnit().

Fiecare strategie de conversie oferă o metodă convert() care primește o valoare de conversie și returnează rezultatul convertit conform strategiei specifice. De asemenea, fiecare strategie oferă și o metodă getUnit() care returnează unitatea de măsură rezultată în urma conversiei.

Prin utilizarea acestui design pattern, se permite definirea unor strategii diferite pentru o anumită funcționalitate, în acest caz, conversia între unități de măsură. Astfel, se obține o separare clară între algoritmul de conversie și implementarea acestuia, permițându-se schimbarea sau adăugarea ușoară a unor noi strategii de conversie fără a modifica codul existent.

În programul dat, strategiile de conversie sunt utilizate în momentul selectării opțiunilor de conversie. În funcție de opțiunea selectată, se creează și se utilizează instanța corespunzătoare a strategiei de conversie pentru a realiza calculele necesare și a obține rezultatul convertit.

**Mediator Design Pattern**

 Figura 1.4. Această diagramă UML prezintă relația între clasele implicate în Mediator Design Pattern.

Clasa "Mediator" definește o interfață comună pentru comunicarea între obiectele "Component" și reține o referință către obiectul "Command" care va fi executat. Clasele concrete "ConcreteMediator" și "ConcreteComponent" implementează interfața "Mediator" și "Component" respectiv.

Relația dintre "Mediator" și "Component" este una de agregare, unde obiectul "Mediator" deține o referință către obiectele "Component" și le coordonează interacțiunea. Există o dependență între "Mediator" și "Command", unde "Mediator" utilizează un obiect "Command" pentru a executa acțiunile specifice.

Această abordare permite decuplarea obiectelor "Component" și le oferă un mod de comunicare indirectă prin intermediul "Mediatorului", eliminând dependențele directe între ele.

# **Concluzie :**

În concluzie, proiectul prezentat demonstrează utilizarea eficientă a diferitelor modele de design, precum Observer, Command, Strategy și Mediator, pentru a implementa o aplicație simplă de conversie a unităților de măsură. Aceste modele oferă o structură organizată și modulară, permit separarea responsabilităților și asigură flexibilitate și extensibilitate în funcționalitatea programului. Aplicația exemplifică bune practici de proiectare și implementare a design patterns, îmbunătățind claritatea, reutilizabilitatea și întreținerea codului.

# **Bibliografie :**

<https://app.diagrams.net/>

<https://refactoring.guru/design-patterns>

<https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm>

<https://www.freecodecamp.org/news/the-basic-design-patterns-all-developers-need-to-know/>