Ministerul Educaţiei, Culturii și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Departamentul Ingineria Software și Automatică

**RAPORT**

Lucrare de laborator Nr.5

Disciplina: Analiza Modelarea Sistemelor

Tema: Studiul şi analiza abstracţiilor OO şi claselor în UML (diagramele de clase).

A efectuat: st.gr. Vlașițchi Ștefan TI-216,

A verificat : asist.univ.   
 Sava Nina

lect.univ.  
Melnic Radu

Chișinău 2023

**Scopul**:Studierea noțiunilor de clasă, atribut și funcție, moștenire, compoziție, agregare, asociere, dependență

**Sarcina:** de realizat 3-4 diagrame de clase pentru sistemul informațional ales

**Considerații teoretice**

Diagrama de clase este una dintre cele mai importante diagrame din proiectarea orientata pe obiecte. Aceasta reprezinta o representare grafica a claselor care sunt utilizate intr-un sistem si a relatiilor dintre ele.

In cadrul unei diagrame de clase, fiecare clasa este reprezentata ca o caseta cu trei sectiuni diferite: numele clasei, atributele si metodele. Atributele sunt variabilele sau proprietatile asociate cu clasa, in timp ce metodele reprezinta comportamentul clasei si ofera functionalitati utilizatorului. Atributele si metodele sunt adesea denumite si membrii de clasa.

Relatiile dintre clase sunt reprezentate prin sageti, care indica directia relatiei si tipul acesteia. Cele mai comune tipuri de relatii sunt relatia de mostenire, de compozitie, de agregare si de dependenta.

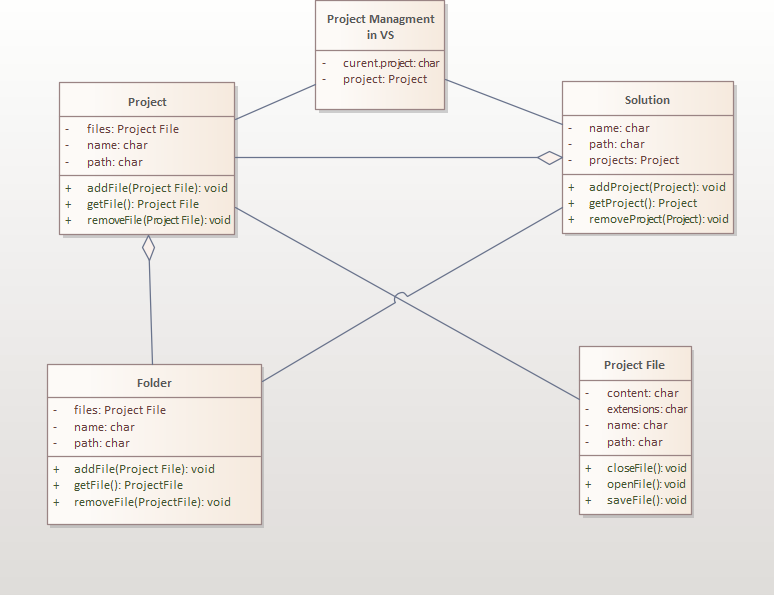
Relatia de mostenire reprezinta o relatie intre doua clase, in care o clasa extinde sau mosteneste atributele si metodele altei clase. Clasa care este mostenita este cunoscuta ca superclasa sau clasa de baza, iar clasa care mosteneste este cunoscuta ca subclasa sau clasa derivata.

Relatia de compozitie reprezinta o relatie intre doua clase, in care o clasa este compusa din una sau mai multe instante ale altei clase. Aceasta relatie indica faptul ca obiectele din clasa compusa nu pot exista fara obiectele din clasa compunatoare.

Relatia de agregare este asemanatoare cu relatia de compozitie, dar in acest caz obiectele din clasa agregata pot exista independent de obiectele din clasa agregatoare.

Relatia de dependenta reprezinta o relatie intre doua clase, in care o clasa utilizeaza serviciile sau atributele altei clase. Aceasta relatie indica faptul ca modificarea clasei furnizoare poate afecta clasa care depinde de ea.

In concluzie, diagrama de clase este o reprezentare grafica a claselor si relatiilor dintre ele intr-un sistem. Aceasta poate ajuta la intelegerea structurii si a interactiunilor dintre componentele unui sistem, precum si la identificarea eventualelor probleme de design si a punctelor critice ale aplicatiei.

**Implementare, rezultate practice:**

***Figura 1.*** Reprezentarea functionalitaii “Project Managment in VS”

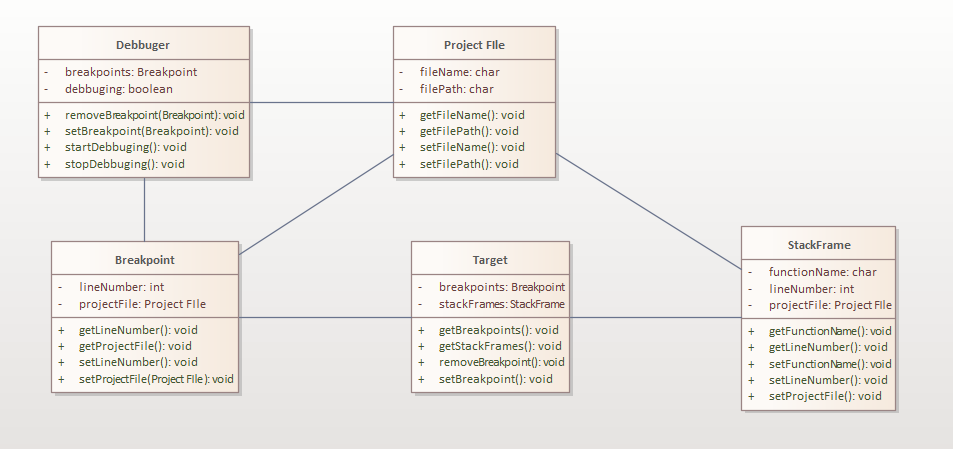
Diagrama arată următoarele clase:

1. Solution - clasa principală care reprezintă o soluție în Visual Studio, care poate conține mai multe proiecte.
2. Project - clasa care reprezintă un proiect în cadrul unei soluții.
3. Folder - clasa care reprezintă un folder în cadrul unui proiect.
4. ProjectFile - clasa care reprezintă un fișier în cadrul unui proiect.

Relațiile între aceste clase sunt următoarele:

1. O soluție poate conține unul sau mai multe proiecte, deci există o relație de asociere între clasa Solution și clasa Project.
2. Un proiect poate conține mai multe foldere, deci există o relație de agregare între clasa Project și clasa Folder.
3. Un folder poate conține mai multe fișiere, deci există o relație de agregare între clasa Folder și clasa ProjectFile.
4. Un proiect poate avea legături cu mai multe fișiere, deci există o relație de agregare între clasa Project și clasa ProjectFile.

În această diagramă, clasa Solution oferă metode pentru crearea și gestiunea proiectelor, inclusiv adăugarea și eliminarea acestora. Clasa Project are metode pentru gestionarea fișierelor și folderelor asociate, precum și pentru compilare și rulare. Clasa Folder și clasa ProjectFile sunt utilizate pentru a organiza și gestiona fișierele în cadrul unui proiect.



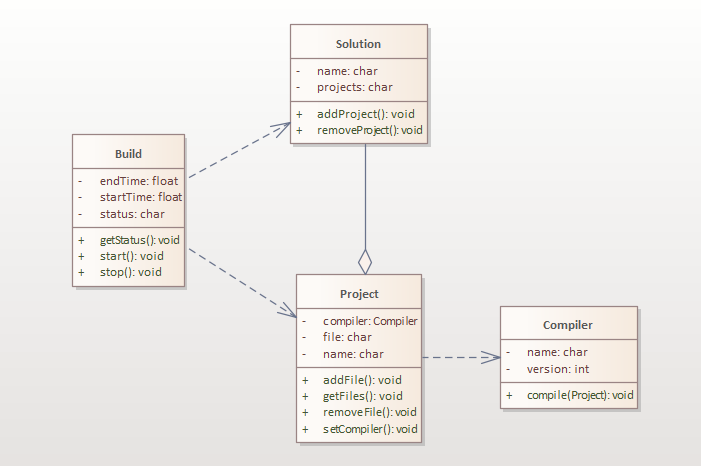
***Figura 2.*** Reprezentare functionalitaii “Debbuging”

Diagrama de mai sus reprezintă o simplificare a arhitecturii unui debugger în cadrul aplicației Visual Studio. Debugger-ul este responsabil pentru găsirea și remedierea problemelor din codul sursă al aplicației.

În această diagramă sunt reprezentate patru clase principale:

1. Debugger - clasa principală a debugger-ului, care are metodele pentru pornirea și oprirea de sesiuni de debug și funcții pentru găsirea simbolurilor și încărcarea modulelor.
2. Breakpoint - clasa care reprezintă un punct de oprire în codul sursă.
3. Target - clasa care reprezintă ținta execuției în cadrul sesiunii de debug. Acesta poate fi o aplicație, un proces sau o altă entitate care este supusă depanării.
4. StackFrame - clasa care reprezintă un cadru de stivă în cadrul apelurilor de funcții și a execuției instrucțiunilor.

Clasa Debugger utilizează obiecte de tip Breakpoint și ProjectFile pentru a controla execuția codului sursă. Clasa Breakpoint utilizează obiecte de tip ProjectFile pentru a identifica locația unde se va face oprirea în execuție. Clasa Target utilizează obiecte de tip Breakpoint și StackFrame pentru a urmări execuția programului. Clasa StackFrame utilizează obiecte de tip ProjectFile pentru a determina locația exactă a execuției.



***Figura 3.*** Reprezentare functionalitaii “Build”

In diagrama din figura 3 este reprezentata diagrama de clase a functionalitatii Build in aplicatia Visual Studio

1. Class "Build": Aceasta clasa reprezinta o operatiune de build in aplicatia Visual Studio.
2. Class "Solution": Aceasta clasa contine informatii despre o solutie Visual Studio, cum ar fi fisierele sursa si configuratia proiectelor.
3. Class "Project": Aceasta clasa contine informatii despre un proiect dintr-o solutie Visual Studio, cum ar fi fisierele sursa, referintele si configuratia.
4. Class "Compiler": Aceasta clasa reprezinta compilatorul folosit pentru a compila proiectele din solutie.

Relatiile dintre aceste clase arata astfel:

1. "Build" are o relatie de dependenta cu "Solution" si "Project", deoarece trebuie sa cunoasca continutul acestora pentru a le compila.
2. "Solution" are o relatie de agregare cu "Project", deoarece contine o colectie de proiecte.
3. "Project" are o relatie de dependenta cu "Compiler", deoarece trebuie sa cunoasca tipul de compilator folosit pentru a compila codul sursa.

**Concluzii:**

Studiul şi analiza abstracţiilor OO şi claselor în UML au relevat faptul că acestea sunt instrumente puternice pentru modelarea şi analiza sistemelor de software. Abstracţiile OO, cum ar fi încapsularea, polimorfismul şi moştenirea, permit dezvoltatorilor să creeze modele mai concise, mai flexibile şi mai uşor de înţeles, ceea ce duce la o dezvoltare mai rapidă şi o mentenanţă mai uşoară a sistemelor de software.Totodata, diagrama de clase reprezinta o metoda eficienta de documentare si pastrare a informatiilor despre sistemul software, permitand astfel o mai mare uscurentare a procesului de dezvoltare si mentenanta.In concluzie, abstractiile OO si diagrama de clase sunt elemente esentiale ale programarii orientate pe obiecte si pot ajuta la crearea unui cod mai modular, mai usor de inteles si de mentinut, ducand la o dezvoltare software mai eficienta si mai calitativa.

**Bibliografie**

1. **Melnic R., Sava N.** Indrumar metodic “Analiza si modelarea sistemelor informationale”.
2. **Introduction to Visual Studio :** <https://www.geeksforgeeks.org/introduction-to-visual-studio/>
3. **Aplicatia GIT:** <https://github.com/>
4. **Crearea unei diagrame de colaborare UML** : <https://support.microsoft.com/ro-ro/office/crearea-unei-diagrame-de-colaborare-uml-6978fee5-3319-4fbf-8a45-7cfd1062a6d4>
5. **Install NuGet client tools** : <https://learn.microsoft.com/en-us/nuget/install-nuget-client-tools>