Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Departamentul Ingineria Software și Automatică

**RAPORT**

Lucrare de laborator Nr.5

Disciplina: Analiza și modelarea sistemelor

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

Au efectuat: st. Popa Cătălin, TI-211

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 o reprezentare grafică a structurii statice a unui model de sistem, care utilizează terminologia claselor programării orientate pe obiecte. Aceasta poate descrie diferite legături între entitățile sistemului, inclusiv obiecte și subsisteme, precum și structura internă și tipurile de relații dintre acestea. Cu toate acestea, diagrama de clase nu conține informații despre aspectele temporale ale funcționării sistemului.În esență, diagrama de clase reprezintă o dezvoltare ulterioară a modelului conceptual al sistemului, care poate include interfețe, pachete, relații și chiar exemple, cum ar fi obiecte și legături. Aceasta este utilizată pentru a înțelege modelul structural static al sistemului și este considerată o reprezentare grafică a legăturilor structurale ale modelului logic al sistemului, care sunt independente și invariante în timp.În limbajul UML, o clasă definește totalitatea de obiecte care au aceeași structură, comportament și relații cu obiectele din alte clase. Grafic o clasă se reprezintă printr-un dreptunghi care poate fi divizat de linii orizontale în secţiuni.

În limbajul UML standardizarea înscrierii atributelor de clasă se supune regulelor sintactice. Fiecărui atribut de clasă îi corespunde rândul textului, care este format din specificatorul de vizibilitate a atributului, numelui lui, tipul sensului şi, posibil sensul final:

«specificatorul de vizibilitate» «numele atributului» [multiplicitate]:

«tipul atributului»=«sensul final» {aliniat-proprietate}

Specificatorul de vizibilitate poate primi unul dintre cele trei sensuri şi concomitent reflectă cu ajutorul simbolurilor speciale:

* Simbolul ,,**+**” înseamnă atributul cu regiunea de vizibilitate de tip public (public). Atributul cu această regiune de vizibilitate poate fi accesat sau văzut din altă clasă de pachet, în care este stabilită diagrama.
* Simbolul ,,**#**” înseamnă atributul cu regiunea de vizibilitate de tip protecţie (protected). Atributul cu această regiune de vizibilitate nu poate fi accesat sau văzut pentru toate clase în afară de subclasele acestei clase
* Simbolul ,,**-**” înseamnă atributul cu regiunea de vizibilitate tipului privat. (private). Atributul cu această regiune de vizibilitate nu poate fi accesat sau văzut pentru toate clasele fără excepţie.

În afară de organizarea internă sau structură claselor în diagrama corespunzătoare sunt indicate diferite relaţii între clase. În urma căruia totalitatea tipurilor astfel de relaţii este fixată în limbajul UML şi este presupusă de semantica astfel tipurilor de relaţii. În limbajul UML relaţiile de bază şi legăturile sunt:

* Relaţia de dependenţă - implică o relație semantică între două elemente modele sau două mulțimi de elemente și arată corespondența dintre două sau mai multe clase.
* Relaţia de asociere - corespunde prezenţei unei relaţii între clase și conectează două sau mai multe clase între ele.
* Relaţia de generalizare - o relaţie taxonometrică între două elemente de acelaşi tip: elementul generalizat (părinte) şi elementul specializat (descendent).
* Relaţia de realizare - relație între două clase, prin care una dintre ele implementează un anumit comportament specificat într-o interfață sau într-o clasă abstractă.

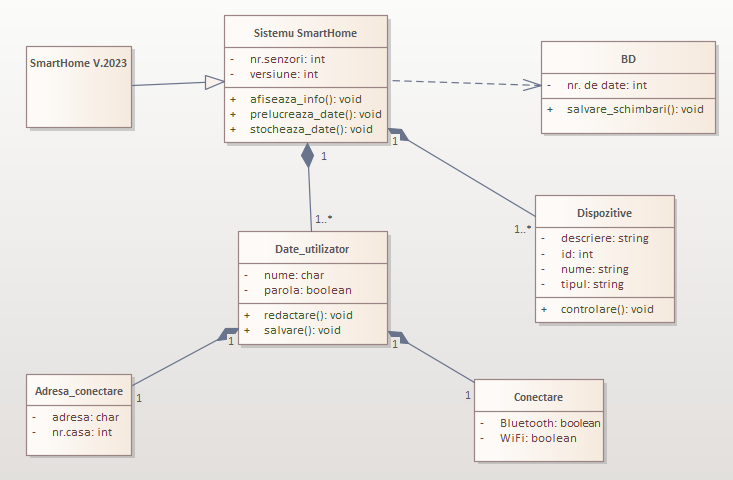
Fiecare dintre aceste relații are reprezentare grafică proprie pe diagramă care reflectă interconexiunele între obiectele claselor corespunzătoare.

Tema pe care trebuie să o implementez eu, se numește **Smart Home**.

Smart Home, este o casă care este echipată cu dispozitive și sisteme electronice integrate care pot fi controlate prin intermediul unor aplicații mobile sau dispozitive de control vocal. Aceste dispozitive sunt proiectate pentru a simplifica și automatiza diferitele aspecte ale vieții cotidinene, cum ar fi iluminatul, încălzirea, aerul condiționat, securitatea, divertismentul și multe altele. Exemple de dispozitive inteligente care pot fi utilizate într-un smart home includ: termostate inteligente, dispozitive de iluminat, camere de securitate, senzori de fum și monoxid de carbon, prize inteligente, asistenți digitali. Aceste dispozitive sunt conectate la o rețea Wi-Fi sau Bluetooth, iar proprietarul casei poate controla și monitoriza aceste dispozitive de la o distanță, folosind o aplicație mobilă sau dispozitivele de control. Un smart home nu numai că oferă confort și ușurință în utilizare, dar poate și economisi bani prin utilizarea eficientă a energiei, gestionarea mai bună a timpului și reducerea riscurilor de securitate. Prețul pentru crearea unui sistem smart home poati varia considerabil, în funcție de dimensiunea casei, numărul de dispozitive și sisteme integrate și complexitatea sistemului. În general, proprietarii de case ar trebui să se aștepte să plătească între câteva mii și câteva zeci de mii de dolari pentru a crea un sistem smart home complet și complex, în funcție de nevoile și preferințele lor. Cu toate acestea, consider că, toate aceste resurse se merită, deoarece într-un final casa ta, devine mult mai funcționabilă și o poți controla de la distață.

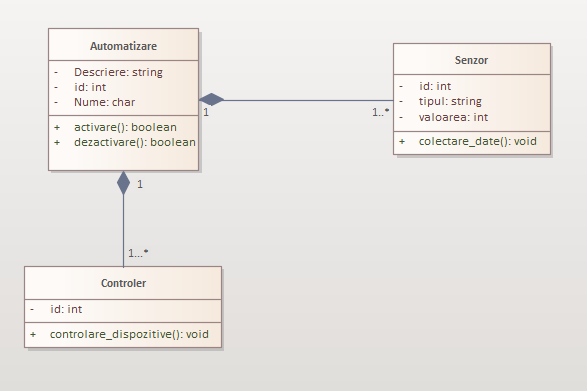
**Diagramele realizate**

Mai întâi voi reprezenta în figura 1, funcționalitatea sistemului SmartHome. Clasa de bază este *Sistemul SmartHome*, care are la evidență câți senzori sunt conectați la el și ce versiune este. Respectiv, el are relații de compoziție cu *Date\_utilizator* și *Dispozitive*. În clasa dispozitive, este prezentă informația despre dispozitiv și funcția de controlare. *Date\_utilizator* conține încă două clase, care reprezintă informații despre adresa de unde este realizată conectarea și metoda de conectare.



**Figura 1.** Principiul de funcționare

În pasul următor, am reprezentat, în figura 2, diagrama de automatizare a procesului de control asupra dispozitivelor. În diagrama respectivă am realizat procesul de automatizare. Respectiv avem clasa automatizare, la care sunt conectate prin relația de compoziție alte două clase *Controler* și *Snezor*. În clasa automatizare avem informații despre dispozitiv, precum și funcția de activare și dezactivare a lui. În clasele controler și senzor este prezentă, de asemenea, informația despre dispozitiv și funcția de control a dispozitivului și de colectare a datelor.



**Figura 2.** Automatizarea.

În figura 3, am realizat diagrama de clase pentru dispozitive. Am luat ca exemplu două dispozitive, camera video și condiționerul. În clasa cameră video, este prezentă informația despre dispozitiv și funcția de activare și oprire a înregistrării. În clasa condiționer de asemenea este informația despre dispozitiv și funcția de setare a temperaturii.

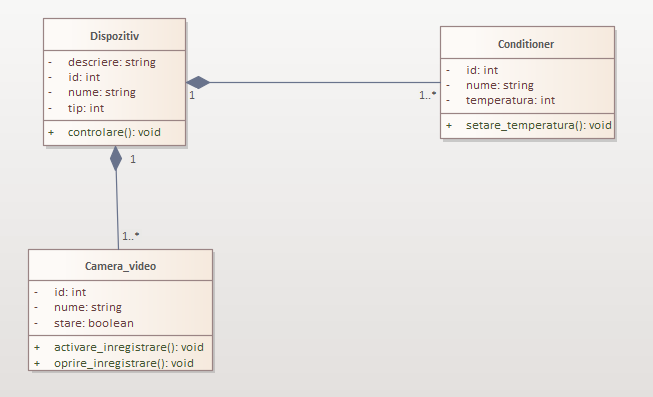


Figura 3. Diagrama de clase pentru dispozitive.

**Concluzii:**

În concluzie, în urma studierii noțiunilor de clasă, atribut și funcție, moștenire, compoziție, agregare, asociere și dependență în programarea orientată pe obiecte, am putut să realizăm 3 diagrame de clasă pentru un sistem informațional al unui sistem Smart Home. Aceste diagrame reprezintă structura statică a sistemului, precum și relațiile dintre diferitele componente ale sale. În cadrul acestor diagrame am putut să învățăm despre moștenirea, compoziția, agregarea, asocierea și dependența dintre clase. Aceste concepte sunt deosebit de utile în dezvoltarea sistemelor complexe, unde relațiile dintre diferitele componente trebuie să fie bine definite și gestionate. Aceste diagrame de clasă pentru un sistem informațional al unui sistem Smart Home ne-au ajutat să aplicăm și să înțelegem mai bine noțiunile de clasă, atribut și funcție, moștenire, compoziție, agregare, asociere și dependență.

**Bibliografie**

**1. Smart Home**, [Sursă electronică]: <https://www.investopedia.com/terms/s/smart-home.asp>.

**2.** **Class diagram**, [Resursă electronică]: [Class diagrams - IBM Documentation](https://www.ibm.com/docs/en/rsm/7.5.0?topic=structure-class-diagrams)