Lucrarea 11: Diagrame UML. Diagrama de componente, diagrama de clase

(Endless cakes! The story that never ends…)

Ce ne propunem astăzi

Azi vom învăța cum să utilizăm diagrame UML (nu toate, doar diagrame de componente și diagrame de clase) pentru modelarea soluțiilor problemelor de Programare Orientată pe Obiecte. Totdeauna, o reprezentare grafică este binevenită, fiind mai sugestivă decât un text descriptiv. Astfel, înțelegem mai bine ce e de făcut și devine mai ușor de scris codul programului. ***CakeMaker*** *este un proiect ce are ca scop trecerea studenților prin toate etapele necesare dezvoltării unui proiect din punctul de vedere al unui programator.*

Obiectivele *ce vor fi urmărite în acest proiect vor fi:*

- *citirea unei arhitecturi de aplicație (**design**);*
- *învățarea unor elemente ale limbajului UML;*
- *utilizarea conceptelor de Programare Orientată pe Obiecte pentru implementarea proiectului;*
- *testarea aplicației (unit testing); (opțional)*

Despre proiect

Aplicaţia simulează un aparat de făcut prăjitură automat cu resurse nelimitate. Rolul lui principal este să servească clientul cu prăjitura dorită. În altă ordine de idei clientul va avea un meniu principal care îi va oferi clientului anumite posibilităţi de selecţie. Vom urmări mai jos arhitectura completă a aplicaţiei noastre.

Rețineți că aplicația simulează crearea de prăjituri la comandă….**nelimitat** ☺

Cum începem o aplicaţie

În primul rând trebuie să știm câte ceva despre UML, cel puțin câteva noţiuni esenţiale. Acestea sunt prezentate la curs (*Cursul 12. UML. Şabloane de proiectare*, pe *Campus Virtual* *https://cv.upt.ro/course/view.php?id=1913* dar puteți citi suplimentar și altele (găsiți foarte multe informații utile pe Internet) spre exemplu la https://ro.wikipedia.org/wiki/Unified_Modeling_Language. Avem nevoie de aceste noţiuni pentru a reuşi mai departe să înţelegem diagramele ce ni se vor prezenta. Iar pe scurt sunt prezentate majoritatea tipurilor de diagrame pe care le putem crea, la https://drive.google.com/open?id=1SyWo53ZL2nPw2RRcDvTMYsB-5nD8FS84.

Acum că noţiunile sunt clarificate, vă voi prezenta pe scurt cu ce diagrame vom lucra în continuare:

- **Diagrama de componente** - prezintă componentele pe care le vom dezvolta în această aplicaţie și modul de interacțiune dintre acestea. Într-un proiect, această diagramă este o privire de ansamblu (un fel de „arhitectură”) și etalează puține

informaţii dar reprezintă o piesă principală. În fond, pentru a pricepe o soluție, trebuie să avem un punct de plecare, o viziune generală, precum cea oferită de această diagramă. O componentă poate să conţină una sau mai multe clase.

- **Diagrama de clase** - descrie componentele în întregime. Ea cuprinde toate caracteristicile şi funcţionalităţile entităţilor unei componente. Mai mult decât atât, această arată şi relaţiile dintre entități (clase). Pornind de la această diagramă, suntem capabili să construim codul sursă al aplicației.

Deci **diagrama de componente** ne permite să înțelegem **cum arată aplicația** – **ca viziune generală**, ca punct de pornire a înțelegerii acesteia, iar **diagrama de clase** ne permite să înțelegem **cum e construită aplicația**, putând astfel chiar să o construim pe bune (cam ca și planul unei case, preluat de zidari !).

Acum să vedem cum facem aplicaţia să scoată produsul foarte plăcut la gust „prăjitură” ☺.

Diagrama de componente este prezentată în *Fig. 1*:

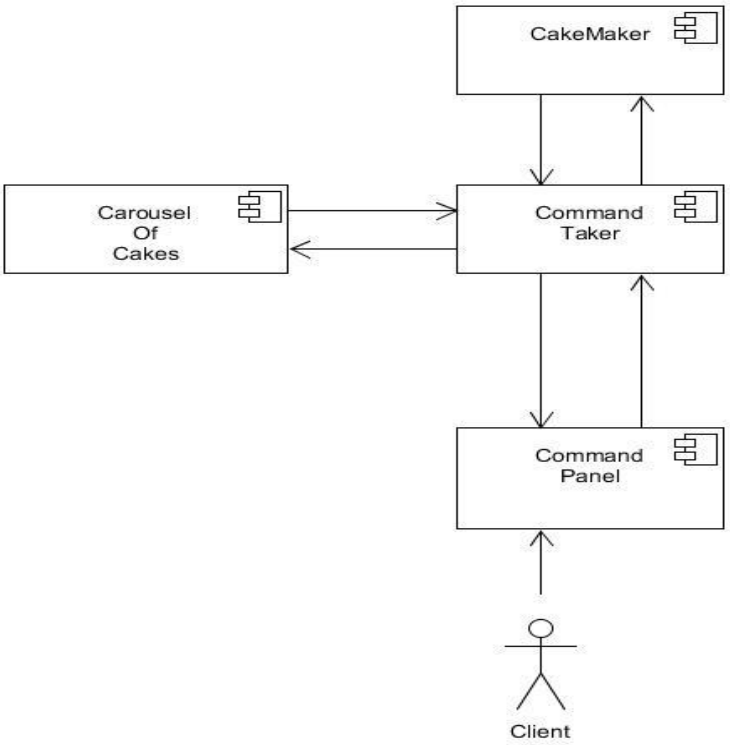

Fig. 1. Diagrama de componente a aplicației CakeMaker

Se identifică următoarele **componente**:

- Command Panel - această componentă este interfaţa cu clientul, dat fiind că este o simulare această va fi CLI (command line interface). El va conţine toate funcţiile necesare pentru a satisface dorinţele clientului.
- Command Taker - această componentă preia comanda de la CommandPanel. El comunică, CakeMaker and CarouselOfCakes.

- Functionalitatea lui CommandTaker este aceea de a onora comenzile date de client. El le administrează conform funcţionalităţii lui, şi anume: în momentul în care primeşte o comandă de la CommandPanel. Onorarea comenzii constă în verificarea cum că produsul se află în CarouselOfCakes (mini storage), dacă aceasta există, va fi eliminat din carousel of Cakes şi îi va distribuii clientului. În cazul în care nu se află în CarouselOfCakes, acesta îl va interoga pe CakeMaker cerându-i să onoreze comanda.

- CarouselOfCakes - acesta este un storage de prăjituri. Are o capacitate maximă de 12 prăjituri. El va accepta doar elemente de tipul Prăjitura. Acest aspect în vom aborda mai târziu în proiect.
- CakeMaker - este cel care face prajitura. El realizează o prăjitură într-un interval specificat in reteta.

Diagrama de clase este prezentată în *Fig. 2*:

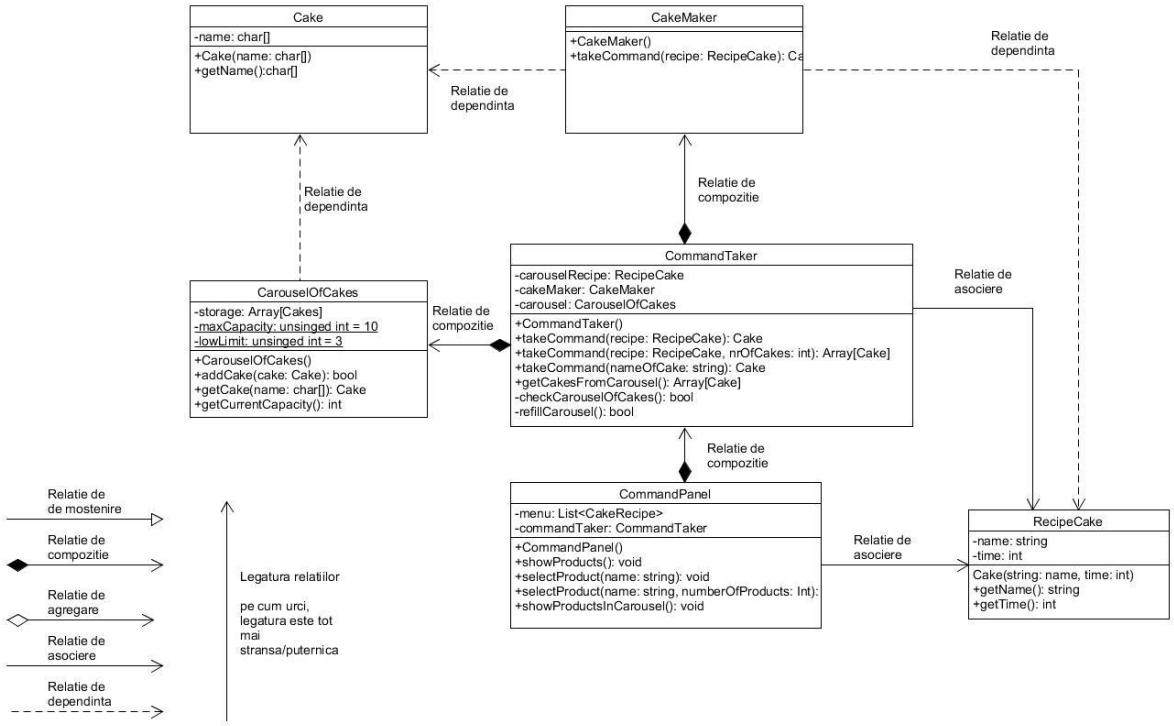

Fig. 2. Diagrama de clase pentru CakeMaker

Diagrama de clase este creată pentru a arăta legăturile dintre entităţi şi a descrie funcţionalităţile şi caracteristicile lor. Pe rând vom lua fiecare clasă şi vom discuta funcționalităţile și atributele fiecăreia:

CommandPanel

- atributele clasei: lista de reţete şi componenta CommandPanel
- functionalităţi:

- **showProducts():** va afişa toate produsele care sunt disponibile pentru cumpărare
- **selectProduct(name: string):** va selecta produsul dorit din meniu şi îl va transmite mai departe lui CommandTaker pentru a fi preluată comanda.
- **selectProduct(name: string, nrOfProducts: int):** supraîncarcată metoda de mai sus fiind precizat şi cantitatea produsului dorit.
- **showProductsInCarousel**(): va afişa în consolă ce va produse sunt în depozitul circular.

CommandTaker

- atributele clasei: **CarouselOfCakes** – cel ce detine cele mai cumparate prajituri, **CakeMaker**-ul cel ce face prijiturile, **CakeRecipe –** lista cu prajituri gata facute.
- funcţionalităţi:
  - **takeCommand(recipe: CakeRecipe): Cake -** preia comanda de la CommandPanel si o trimite mai departe CakeMaker-ului. Inainte El mai face o verificare daca prajitura se afla in CarouselOfCakes, dacă acolo se află produsul dorit atunci se va lua direct de acolo.
  - **takeCommand(recipe: CakeRecipe, nrOfCakes): Array[Cake] -** preia comanda în cazul în care sunt mai multe. Dacă se cere o cantitate de produse atunci se va cere direct CakeMaker-ului.
  - **getCakesFromCarousel(): Array[Cake] -** returnează lista de prăjituri ce se află în CarouselOfCakes.
  - **checkCarouselOfCakes(): bool** - verifică capacitatea curentă din CarouselOfCakes.
  - **refillCarousel():void** - dacă capacitatea din Carousel este mică, această metodă îi va cere lui CakeMaker să facă atâtea prăjituri câte sunt necesare pentru a umple CarouselOfCakes la maxim.

CakeMaker

- atributele clasei: -
- funcţiile clasei:
  - **takeCommand(recipe: RecipeCake): Cake -** această metodă simulează crearea unei prăjituri şi anume pentru fiecare reţetă primită, această aşteaptă 5 secunde, după care ajunge să returneze un obiect de tip Cake. Din reţetă va prelua numele.

Partea practică. Mod de lucru:

- Studiaţi diagramele şi specificaţiile acestora. Toate întrebările scrieţile pe o foaie şi încercaţi să vedeţi la final care ar fi teoretic scopul proiectului.
- Creaţi un proiect empty, iar pe acesta veţi lucra în felul următor. Prima dată creaţi doar clasele, fără atribute sau funcţionalităţi, doar clasele cu constructorii necesari şi legăturile dintre fișiere (cine pe cine include).
- Vă asiguraţi că proiectul este compilat.
- Acum e momentul ca în fiecare clasă să adăugaţi atributele necesare.

- Vă asiguraţi că proiectul este compilat.
- Adăugăm şi definiţiile funcţiilor (dar fără implementare, adică fără conținut).
- Bineînţeles, ne asigurăm că proiectul este compilat.
- Adăugăm şi implementările funcţiilor, iar pe măsură ce ajungem să facem implementarea, din **Main.cpp**, apelăm metode care să asigure funcționalitățile proiectului, conform cerințelor inițiale.
- În acest fel ajungem să creăm un proiect de la zero, bazat pe specificaţii, urmărind concepte de Programare Orientată pe Obiecte, utilizând limbaj UML (foarte important) şi într-un final ajungem să avem bineînţeles **prăjituri**! (deocamdată doar simulate, dar dacă dvs. mai adăugați un pic de pasiune, puteți să faceți mașinăria pe bune, oricum deja o parte din software ar fi gata !)

Cu ce ne-am ales ?

Însuşirea utilizării UML este una din cele mai avansate aspecte specifice activității de Proiectare Orientată pe Obiecte. Având cunoștințe despre UML vom fi capabili să avem o foarte bună productivitate a dezvoltării de aplicații Orientate pe Obiecte și pe baza acestor cunoștințe să accedem la utilizarea Șabloanelor de Proiectare, care ne permit o creștere și mai mare a productivității dezvoltării de aplicații software. Astfel, aplicații care ar necesita dezvoltare de luni de zile pot fi realizate cu succes în câteva zile, ceea ce e de natură să motiveze interesul nostru pentru UML.