Proiectul 2

Experiment and build your portfolio

# Grupa 143

## Exercițiul 1 – Console Forms

Formulare de feedback sau evaluare, de tipul Google Forms

Mai exact, vom implementa formulare cu aceiasi funcitonalitate precum Google Forms, dar cu diferenta ca vor fi rulate in consola.

### Clasele

Pentru a utiliza cât mai util programarea orientată pe obiecte, vom avea clase clare si simple, care au scopurile la care ne-am aștepta:

* **User** – reține denumirea utilizatorului, dar și un id unic
* **FormInput** - va avea diverse forme, precum **MultipleChoice**, **ShortAnswer**, **LinearGradient**, **CheckBox**, **MultilineAnswer**.
* **Form** – va semnifica un formular care poate avea toate tipurile de întrebări menționate mai sus, dar și formulare cu punctaj care pot avea doar opțiuni precum **MultipleChoice**, **LiniarGradient** sau **CheckBox; Nu** vom putea crea formulare de tipul Form. (abstract class)
* **GradedForm –** reprezintă un formular care conține doar întrebări de tipul MultipleChoice, LinearGradient sau CheckBox**. TODO de scris despre fiecare**
* **FeedbackForm –** un formularcare poate conține orice tip de input-uri
* Fiecare Graded form va conține și un barem **TODO**
* **FormCreator** – va fi o clasă a cărei rol este crearea pas cu pas a unui formular, prin intermediul unui meniu interactiv TODO.

## Demonstrație

TODO video

### Simplitatea codului

Codul din funcția main (a.k.a. the driver code) va fi extrem de simplu:

Pentru rularea

*int* main() {  
 Game game; *// nu are nevoie de date,   
 // deoarece jucatorii si scorurile le citim de la tastatura* game.mainLoop();  
}

### Cerințele

* Jocul va începe întotdeauna prin citirea datelor celor doi jucători. Datele precum comenzile nu vor fi citite de la tastatură, ci numai denumirea jucătorilor.
* Există date care vor fi inițializate la fiecare nou joc (scorul, numele jucătorilor), dar și update-uri la nivelul **HighscoreTable** și **PlayerUi** (în sensul că PlayerUi ar trebui resetat, iar HighscoreTable resetat cu numele și id-ul player-ului).

## Exercițiul 2 – Pharmacy DBMS

*--*

## Exercițiul 3 – 2048 Game

*Implementing a Classical Game*

***For this game, you won’t need multithreading.***

# Grupa 152

## Exercițiul 1 - Pong

*Implementing a Classical Game*

***For this exercise you’ll make use of multithreading and we’ll provide an example.***

### Clasele

Pentru a utiliza cât mai util programarea orientată pe obiecte, vom avea clase clare si simple, care au scopurile la care ne-am aștepta:

* **Player** – reține denumirea playerului, dar și un id unic
* **PlayerUI** – aici vom implementa multithreading. Mai exact, PlayerUI este o clasă care va citi comanda unui player pentru a putea fi executată de către GameScreen
* **GameScreen** – această se ocupă de afișarea stării jocului. Conține date despre numele jucătorilor, scorul pe care îl pot câștiga (crește la fiecare lovire a mingii), mărimea tablei și poziția mingii
* **Highscore** – conține scorul (un număt natural), dar și id-ul jucătorului
* **HighscoreTable –** conține o listă de scoruri
* **Game** – Jocul va conține doi jucători, o interfață pentru fiecare, un ecran al jocului și un table de scoruri

### Demonstrație

### Simplitatea codului

Codul din funcția main (a.k.a. the driver code) va fi extrem de simplu:

*int* main() {  
 Game game; *// nu are nevoie de date,   
 // deoarece jucatorii si scorurile le citim de la tastatura* game.mainLoop();  
}

### Cerințele

* Jocul va începe întotdeauna prin citirea datelor celor doi jucători. Datele precum comenzile nu vor fi citite de la tastatură, ci numai denumirea jucătorilor.
* Există date care vor fi inițializate la fiecare nou joc (scorul, numele jucătorilor), dar și update-uri la nivelul **HighscoreTable** și **PlayerUi** (în sensul că PlayerUi ar trebui resetat, iar HighscoreTable resetat cu numele și id-ul player-ului).

### Cerințe speciale – indicii pentru dezvoltarea jocului

În primul rând, să imaginăm faptul că orice ecran este format din pixeli. În cazul nostru, putem implementa un Pong în care fiecare caracter afișat în consolă **este un pixel**. Spre exemplu, **@** poate reprezenta mingea, **|** o parte a paletei unui jucător, iar **\*** un caracter din conturul jocului:

\* \* \* \* \* \* \* \* \* \*  
\* \*  
\* | \*  
\* @ | \*  
\* | | \*  
\* | \*  
\* | \*  
\* \*  
\* \* \* \* \* \* \* \* \* \*

Cel mai sigur mod de a implementa un joc este cu ajutorul semnalelor și al funcțiilor care le tratează (callbacks apelate când starea tablei sau a unor variabile de stare a jocului sunt modificate, câteodată numite slots, handlers etc)

Vă puteți gândi care sunt datele a căror modificare va determina reafișarea (denumită și redesenare) a întregului joc.

Desigur, pentru a executa redesenarea, este nevoie să „capturăm” semnalele. Ce modalități am avea de a transmite către GameScreen (clasa care va redesena jocul) faptul că tabla de joc s-a modificat?

**Dacă nu vă doriți încă hint-uri, NU treceți la pagina următoare !!!**

Salutare. Urmează mai multe hint-uri.

Pentru a împărți, chiar dacă într-un mod destul de basic, informația că jocul a avut un update (e.g. mingea s-a mișcat, sau un jucător a mișcat paleta sa), este ca obiectul GameScreen și cele 2 obiecte de tipul PlayerUi să aibă în comun un shared\_ptr (de aici denumirea de shared), eventual atât paleta, cât și o variabilă care spunea că a avut loc un update.

PlayerUi va modifica pozitia paletei și va seta variabila de update a poziției, iar GameScreen va deduce din valoarea variabilei de update că trebuie sa modifica tableta astfel încât jocul să se updateze.

## Exercițiul 2 – Cloning Bolt

*An application that simulates bolt while making use of graph theory.*

## Exercițiul 3 – Unsupervised Learning

*==*

## Exercițiul 4 – Cards Game

*==*