Univeritatea “Dunărea de Jos” , Galați

Facultatea de Automatica, Calculatoare, Ingineria Electrică și Electronică

Specializarea Calculatoare

PROIECT

Ingineria programelor

Tema: Butterfly Kyodai

**Student:** Osanu Ioan-Aurel

**Grupa:** 22C32B

**Anul:** III

Introducere:

**P**rogramarea **O**rientata pe **O**biecte (POO)sau **O**bject **O**riented **P**rogramming (OOP)  – este o paradigma de programare care foloseste concepte abstracte (sub forma de clase si obiecte) pentru a crea modele bazate pe elemente din lumea reala. O aplicatie construita pe programare orientata pe obiecte foloseste mai multe obiecte care comunica prin mesaje specifice pentru a rezolva diverse servicii (ale aplicatiei) solicitate.

Aceste **obiecte** sunt capabile sa trimita mesaje (date), sa primeasca mesaje (date) si sa proceseze date. **Scopul programarii orientate pe obiecte este de a incerca sa creasca flexibilitatea si mentenabilitatea programelor**. Deoarece programele create folosind o un limbaj orientat pe obiecte (cum este [Java](https://ramonnastase.ro/blog/tutorial-de-programare-in-java-pentru-incepatori-invata-programare/" \t "_blank)) sunt modulare, ele pot fi mai usor de dezvoltat si mai usor de inteles dupa dezvoltare.

Tema jocului: Butterfly Kyodai;

În continuare, se va crea un joc asemănător cu Butterfly Kyodai utilizând șabloane de design.

Link către jocul Butterfly Kyodai:

<https://www.funnygames.ro/joc/butterfly_kyodai.html>

Analiza jocului:

Jocul permite selectarea a 2 itemi identici dintr-o multitudine de itemi, folosind un anumit algoritm pentru identificarea unui traseu „liber”. Selectarea a doi itemi nu se poate face pe diagonală, ci doar pe orizontală și verticală. Se poate găsi traseu liber si pe marginea aceleiași laturi.

Captura din joc:



Șabloane utilizate:

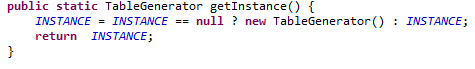
1. Șablonul Singleton

**Singleton** este un model de design creațional care vă permite să vă asigurați că o clasă are o singură instanță, oferind în același timp un punct de acces global la această instanță.

Aceasta clasa declara metoda statica *getInstance* care returneaza aceeasi instanta a aceleasi clase.

Constructorul Singleton ar trebui sa fie ascuns de codul clientului. Apelarea metodei *getInstance* ar trebui sa fie singura modalitate de a obtine obiectul Singleton.

În proiectul meu am folosit acest șablon în clasa TabelGenerator



1. Șablonul Command

Command este un model de design comportamental care incapsuleaza cereri ca obiecte. Acesta declara interfata pentru executarea operatiei.

În proiect este utilizat in clasele Algorithm, GUI si TableGenerator.

Clasa Algorithm este importanta pentru ca contine algoritmul ce are nevoie de mai multe campuri si prelucrari reprezentand exemplul perfect de necesitate a sablonului command.

1. Șablonul Iterator

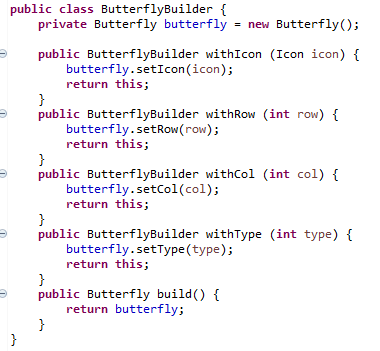
**Iterator** este un model de proiectare comportamentală care vă permite să traversați elemente ale unei colecții fără a expune reprezentarea sa de bază (listă, stivă, copac etc.).

**Acest șablon a fost implementat in clasa ArrayMatrixIterator care parcurge matricea de fluturi.**

1. Șablonul Builder

**Builder** este un model de creație care vă permite să construiți obiecte complexe pas cu pas. Modelul vă permite să produceți diferite tipuri și reprezentări ale unui obiect folosind același cod de construcție.

In proiect, acest sablon este folosit in clasa ButterflyBuilder care modifica campul butterfly, returneaza this si la sfarsit returneaza campul butterfly modificat.



1. Șablonul Facade

Facade este un model de proiectare structurală care oferă o interfață simplificată către o bibliotecă, un cadru sau orice alt set complex de clase.

In proiect, clasa GUI este prin definitie un facade pentru că ascunde in spatele unei interfete grafice toata functionalitatea aplicatiei.

1. Șablonul Composite

Composite este un model de proiectare structurală care vă permite să compuneți obiecte în structuri de copac și apoi să lucrați cu aceste structuri ca și cum ar fi obiecte individuale.

Main-ul este un composite care combina doua clase foarte complexe

* TableGenerator
* GUI

1. Șablonul Mediator

**Mediator** este un model de proiectare comportamentală care vă permite să reduceți dependențele haotice dintre obiecte. Modelul restricționează comunicațiile directe între obiecte și le obligă să colaboreze doar printr-un obiect mediator.

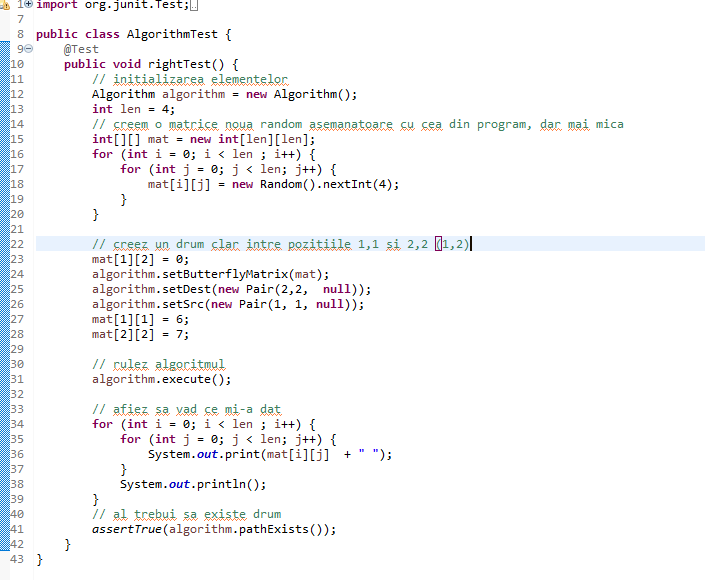
MyPanel este componenta ce mediaza interactiunea dintre aplicatia propiu zisa ( Algorithm ) si interfata grafica ( GUI ).

Această clasa reprezinta tabelul de fluturi.

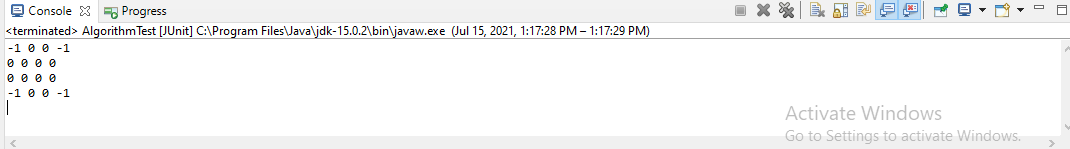
Aceasta are un GridLayout de 10/10 ce reprezinta un tabel de fluturi am folosit de asemenea butterflyArrayMatrixIterator ce este un Iterator design pattern pentru a adauga la fiecare buton nou creat fluturele din matrice cu ajutorul TableGenerator.

Plan de testare

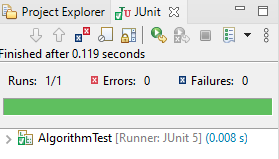
Pentru testare am folosit JUnit pentru a verifica pe o matrice mai mică dacă se găsește drum liber pentru compatibilitatea a doi fluturi.



Rezultatul afisat in consola este urmatorul:



Rezultatul testului este pozitiv și lipsit de erori:



Bibliografie

1. <https://refactoring.guru/design-patterns/catalog>
2. <https://www.javatpoint.com/design-patterns-in-java>
3. Cursurile de la Ingineria Programelor