PLAN NUMERE COMPLEXE - LAB 4-6

Paul Tal

LISTA FUNCȚIONALITĂȚI

F1. Adaugă număr complex la sfârșitul listei **F2.** Inserează număr complex pe o poziție dată **F3.** Șterge element de pe o poziție dată **F4.** Șterge elementele de pe un interval de poziții **F5.** Înlocuiește toate aparițiile unui număr complex **F6.** Tipărește partea imaginară pentru un interval de poziții **F7.** Tipărește numerele cu modulul mai mic decât 10 **F8.** Tipărește numerele cu modulul egal cu 10 **F9.** Suma numerelor dintr-o subsecvență **F10.** Produsul numerelor dintr-o subsecvență **F11.** Lista sortată după partea imaginară **F12.** Filtrare partea reală primă **F13.** Filtrare după modul **F14.** Undo ultima operație

PLANUL DE ITERAȚII

Iterația	Funcționalități
I1	F1, F7, F8 + UI de bază
I2	F2, F3, F6, F9
I3	F4, F5, F10-F14

MODELAREA ITERAȚIEI 1

F1 - Adăugare număr

Utilizator	Program	Descriere
Start aplicație	Meniu principal	Testare + meniu
Selectează “1”	Prompt input	“Introduceti numarul complex”
Tastează “3+4i”	“Numar adaugat: 3+4i modul = 5.000”	Confirmare
Selectează “2”	Lista: “0. 3+4i modul = 5.000”	Afișare listă

F7 - Filtrare modul < 10

Input	Output
Lista: [3+4i, 6+8i, 1+1i]	Afișare numere cu modul < 10
	“3+4i modul = 5.000”
	“1+1i modul = 1.414”
	“Numere excluse: 6+8i modul = 10.000”

F8 - Filtrare modul = 10

Input	Output
Lista: [6+8i, 3+4i]	Căutare modul = 10
	“6+8i modul = 10.000”
	“Sugestii: 6+8i, 10+0i, 0+10i (dacă goală)”

MODELAREA ITERAȚIEI 2

F2 - Inserare la poziție

Utilizator	Program	Descriere
Lista: [3+4i, 6+8i]	Afișare listă curentă	Context pentru inserare
Selectează “6”	Prompt poziție	“Introduceti pozitia pentru inserare (0-2)”
Tastează “1”	Prompt număr	“Introduceti numarul complex”
Tastează “2+3i”	“Numar 2+3i inserat la pozitia 1”	Confirmare
	Lista: [3+4i, 2+3i, 6+8i]	Lista actualizată

F3 - Ștergere de la poziție

Utilizator	Program	Descriere
Lista: [3+4i, 2+3i, 6+8i]	Afișare listă curentă	Context pentru ștergere
Selectează “7”	Prompt poziție	“Introduceti pozitia pentru stergere (0-2)”
Tastează “1”	“Numar 2+3i sters de la pozitia 1”	Confirmare
	Lista: [3+4i, 6+8i]	Lista actualizată

F9 - Suma subsecvenței

Utilizator	Program	Descriere
Lista: [3+4i, 6+8i, 1+1i]	Afișare listă curentă	Context pentru sumă
Selectează “8”	Prompt interval	“Introduceti indexul de start/sfarsit”
Tastează “0”, “2”	Numere: 3+4i, 6+8i, 1+1i	Afișare numere incluse
	Suma: 10+13i modul = 16.401	Rezultat final

TASK-URI ITERAȚIA 1

T1. Clasa ComplexNumber
- Constructor cu real, imaginary
- Metode: get_real(), get_imaginary(), get_string(), get_module()

T2. Calcularea modulului
- Formula: $\sqrt{(a^2 + b^2)}$
- Teste cu diverse cazuri

T3. Storage-ul
- Clasa Storage pentru gestionarea listei
- Metode: append_number(), get_numbers(), filtrări

T4. Parsarea input-ului
- Recunoaștere formate: “a+bi”, “a-bi”, “a”, “bi”, “i”
- Validare

T5. UI și meniu principal
- Meniu cu opțiuni
- Gestionare input/output

T6. Integrare și testare

- Test runner automat

TASK-URI ITERAȚIA 2

T7. Operații aritmetice ComplexNumber
- Metoda add() pentru adunarea numerelor complexe
- Teste pentru operațiile aritmetice

T8. Operații Storage avansate
- insert_number_at_position() cu validare bounds
- delete_number_at_position() cu validare bounds
- sum_numbers_interval() pentru calculul sumei

T9. UI pentru operații noi
- get_position_input() cu validare
- show_sum_result() pentru afișarea rezultatelor
- Actualizare meniu cu opțiuni 6, 7, 8

T10. Menu handlers pentru iterația 2
- handle_insert_at_position()
- handle_delete_at_position()
- handle_sum_subsequence()

T11. Testare completă iterația 2
- Teste pentru toate funcționalitățile noi
- Integrare cu test runner existent

TESTE

ComplexNumber

Input	Output așteptat
ComplexNumber(3, 4)	get_string() = “3+4i”
ComplexNumber(5, −3)	get_string() = “5-3i”
ComplexNumber(3, 4)	get_module() = 5.0
n1.add(n2) cu n1(3,4), n2(5,2)	ComplexNumber(8, 6)

Storage

Operație	Rezultat așteptat
append_number(3+4i)	Lista: [3+4i]
get_numbers_module_less_than(10)	Returnează [3+4i, 1+1i]
get_numbers_module_equal(10)	Returnează [6+8i]
insert_number_at_position(1, 2+3i)	Lista: […, 2+3i la poziția 1, …]
delete_number_at_position(1)	Returnează numărul șters
sum_numbers_interval(0, 2)	Suma tuturor numerelor din interval

Parser Input

Input string	Complex rezultat
“3+4i”	ComplexNumber(3, 4)

“5-2i”	ComplexNumber(5, −2)
“7”	ComplexNumber(7, 0)
“3i”	ComplexNumber(0, 3)
“i”	ComplexNumber(0, 1)

ARHITECTURA

```
main.py
├── test_runner.py
├── app.py
├── ui.py (opțiuni 1-8, h, 0)
├── menu_handler.py (8 handlere funcționale)
├── numar_complex.py (ComplexNumber cu add())
└── storage.py (Storage cu insert/delete/sum)
```

MODELAREA ITERAȚIEI 3

F4 - Ștergere interval de poziții

Utilizator	**Program**	**Descriere**
Lista: [3+4i, 2+3i, 6+8i, 1+1i]	Afișare listă curentă	Context pentru ștergere interval
Selectează “9”	Prompt interval	“Introduceti intervalul pentru stergere”
Tastează “1”, “3”	“Sterse 3 numere din intervalul [1,3]”	Confirmare
	Lista: [3+4i]	Lista actualizată

F5 - Înlocuire toate aparițiile

Utilizator	**Program**	**Descriere**
Lista: [3+4i, 3+4i, 6+8i]	Afișare listă curentă	Context pentru înlocuire
Selectează “10”	Prompt număr vechi	“Introduceti numarul de inlocuit”
Tastează “3+4i”	Prompt număr nou	“Introduceti numarul nou”
Tastează “2+2i”	“Inlocuite 2 aparitii ale lui 3+4i cu 2+2i”	Confirmare

F10 - Produs subsecvenței

Utilizator	**Program**	**Descriere**
Lista: [2+i, 3+4i, 1+i]	Afișare listă curentă	Context pentru produs
Selectează “11”	Prompt interval	“Introduceti intervalul pentru produs”
Tastează “0”, “2”	Numere: 2+i, 3+4i, 1+i	Afișare numere incluse
	Produs: −5+17i modul = 17.720	Rezultat final

F11 - Sortare după partea imaginară

Utilizator	**Program**	**Descriere**
Lista: [3+4i, 1+1i, 6+8i]	Lista nesortată	Context pentru sortare

Selectează “12”	Lista sortată descrescător	Sortare după imaginara
	6+8i modul = 10.000	Imaginara = 8
	3+4i modul = 5.000	Imaginara = 4
	1+1i modul = 1.414	Imaginara = 1

TASK-URI ITERAȚIA 3

T12. Operații aritmetice extinse ComplexNumber
- Metoda multiply() pentru înmulțirea numerelor complexe
- Metoda equals() pentru compararea numerelor complexe
- Teste pentru operațiile aritmetice noi

T13. Operații Storage pentru iterația 3
- delete_numbers_interval() pentru ștergerea unui interval
- replace_all_occurrences() pentru înlocuirea tuturor aparițiilor
- product_numbers_interval() pentru calculul produsului
- get_sorted_by_imaginary() pentru sortarea după partea imaginară

T14. Filtrări avansate Storage
- is_prime() funcție utilitară pentru verificarea numerelor prime
- filter_real_part_prime() pentru filtrarea după partea reală primă
- filter_by_module() pentru filtrarea după modul cu operatori

T15. Sistem Undo cu Delta Tracking (Git-style)
- Clasa UndoManager pentru gestionarea istoricului cu snapshot-uri complete
- Clasa DeltaUndoManager pentru gestionarea istoricului cu delta-uri (git-style)
- DeltaUndoManager stochează doar modificările necesare pentru undo
- save_state() și undo_last_operation() în Storage cu suport dual
- Tracking manual al modificărilor cu operații inverse

T16. UI pentru operații iterația 3
- get_operator_input() pentru alegerea operatorului de comparație
- show_product_result() pentru afișarea rezultatului produsului
- Actualizare meniu cu opțiuni 9-15

T17. Menu handlers pentru iterația 3
- handle_delete_interval()
- handle_replace_all_occurrences()
- handle_product_subsequence()
- handle_sort_by_imaginary()
- handle_filter_prime_real()
- handle_filter_by_module()
- handle_undo()

T18. Testare completă iterația 3
- Teste pentru toate funcționalitățile noi
- Teste pentru sistemul de undo
- Integrare cu test runner existent

ARHITECTURA FINALĂ

```
main.py
├── test_runner.py
├── app.py
├── ui.py (opțiuni 1-15, h, 0)
├── menu_handler.py (15 handlere funcționale)
├── numar_complex.py (ComplexNumber cu add(), multiply(), equals())
└── storage.py (Storage cu toate operațiile + dual undo system)
    ├── UndoManager (snapshot-based)
    └── DeltaUndoManager (git-like delta tracking)
```

SISTEMUL DELTA UNDO

Principiu de Funcționare

DeltaUndoManager implementează un sistem de undo similar cu Git, care stochează doar modificările (delta-urile) în loc de snapshot-uri complete.

Operație	Delta Stocat pentru Undo
append(number)	Tipul: “append”, Numărul adăugat
insert(pos, number)	Tipul: “insert”, Poziția, Numărul
delete(pos)	Tipul: “delete”, Poziția, Numărul șters
delete_interval(start, end)	Tipul: “delete_interval”, Intervalul, Lista numerelor
replace_all(old, new)	Tipul: “replace_all”, Numărul vechi, nou, Pozițiile
filter operations	Tipul: “filter”, Lista (poziție, număr) eliminată

Avantaje Delta vs Snapshot

- Memorie: Doar modificările în loc de copii complete
- Performanță: Mai rapid pentru liste mari
- Scalabilitate: Constant space per operație
- Precisie: Undo exact cu operații inverse

TESTE ITERAȚIA 3

ComplexNumber extinse

Operație	Rezultat așteptat
n1.multiply(n2) cu n1(2,1), n2(3,4)	ComplexNumber(2, 11)
n1.equals(n2) cu n1(3,4), n2(3,4)	True
n1.equals(n2) cu n1(3,4), n2(3,5)	False

Storage iterația 3

Operație	Rezultat așteptat
delete_numbers_interval(1, 3)	Șterge elementele de la poziția 1 la 3
replace_all_occurrences(3+4i, 2+2i)	Înlocuiește toate aparițiile
product_numbers_interval(0, 2)	Produsul numerelor din interval
get_sorted_by_imaginary(True)	Lista sortată descrescător după imaginara
filter_real_part_prime()	Elimină numerele cu partea reală primă

filter_by_module(“<”, 10)	Elimină numerele cu modul < 10

DeltaUndoManager

Operație & Undo	**Rezultat așteptat**
append → undo	Adaugă element apoi îl elimină
insert(pos) → undo	Inserează la poziție apoi elimină de la poziție
delete(pos) → undo	Șterge de la poziție apoi restaurează la poziție
delete_interval → undo	Șterge interval apoi restaurează toate elementele
replace_all → undo	Înlocuiește toate apoi restaurează valorile originale
filter → undo	Filtrează elememente apoi le restaurează la pozițiile originale
limite istoric	Păstrează maximum 10 operații în istoric