Ciudin Antonela 325CB
Programare orientata pe obiecte. Tema 1

Cerinta 1:

Pentru inceput am creat o clasa “Pasager”. Apoi clasele: “Family” si “Group”, care extind prima clasa(pentru ca ambele contin mai multi pasageri). Fiecare din aceste doua clase au cate un vector cu elemente de tip “Pasager”, care primesc parametrii corespunzatori (id, nume etc.). In clasa “Main”, cu metoda main am creat trei vectori: vectori de pasageri singuri, de familii si de grupuri. Primul “for” din main citeste toti pasagerii si ii insereaza mai intai in acesti vectori, pentru o structurare mai buna.

Cerinta 2:

Pentru coada am creat clasa “Queue”, care contine toate metodele aplicate asupra cozii. In main citesc comanda (insert, list, embark sau delete), daca e cazul in functie de id utilizez vectorul corespunzator de entitati(pasageri singur, familii sau grupuri), apoi apelez metoda din clasa “Queue”.

- ⇨ insert(Pasager p, int priority):
  Dupa ce calculez prioritatea entitatii apeland metoda “calcPriority” din main, apelez “insert” din “Queue”, care insereaza initial pe ultima pozitie iar dupa il muta in fucntie de prioritate dupa metoda heap-ului si seteaza prioritatea. Inserarea in coada are loc de pe pozitia 1, pentru a accesa mai usor fiii.(in caza ca e grup sau familie, prioritatea se seteaza pentru prima persoana, fiind aceeasi).

- ⇨ embark():
  Metoda respectiva sterge primul element al cozii, apoi sorteaza coada incepand cu pozitia 1.

- ⇨ list():
  Din list() se apeleaza metoda “printPreorder” care agiseaza elemente sortate din heap in mod root->left->right.

Cerinta 3:

Pentru citirea din sifier am folosit “Scanner”-ul. Am citit intr-un for datele despre pasageri, apoi intr-un while comenzile pana la capatul fisierului. In token stocam cate o linie, apoi accesam elemente ca dintr-un tablou.

Cerinta 4:

Pentru afisare creez in “Queue” un fisier output, care il inchis in main.

Metode suplimentare:

- tipTicket -> determina prioritatea in functie de tipul biletului si intoarce valoarea respectiva.
- age -> determina prioritatea in functie de varsta pasagerului.
- calcPriority -> calculeaza si retuneaza prioritatea pentru un pasager/familie/grup.
- parent -> returneaza parintele elementului primit ca parametru.
- swap -> schimba cu locurile cele doua elemente din coada, primite ca paramentri.
- getPriority -> intoarce prioritatatea in functie de tipul entitatii primite ca parametru, utilizand operatorul instanceof.
- getID -> functioneaza la fel ca getPriority, doar ca returneaza id-ul entitatii.
- sefPriority -> seteaza prioritatea in functie de tipul entitatii.
- sort -> sorteaza coada incepand cu pozitia primita ca parametru. Se verifica initial daca ambii fii exista. Daca exista ambii fii, iar apoi unul din ei are prioritatea mai mare decat perintele, atunci se interschimba (in prioritate fiind fiul stang) iar apoi se continua sortarea pe fiul respectiv. Daca nu exista ambii atunci se verifica aparte pentru fiecare existenta acestuia si se interschimba cu parintele daca este cazul.

Bonus:

- delete():
  In main, cand citesc comanda, stabilesc ce tip de delete este (cu nume sau fara). Daca nu contine nume, atunci apelez metoda “delete” din “Queue”. Se apeleaza metoda “findPasager” care intoarce pozitia entitatii cu id-ul dat ca parametru. Se elimina entitatea si dupa se sorteaza incepand cu pozitia elementului sters. Daca comanda are nume, atunci determin ce reprezinta entitatea: familie sau grup (in cazul pasagerului singur am adaugat o conditie la if-ul pentru comanda fara nume, deoarece se va efectua in mod similar). Apoi apelez metoda din Clasa corespunzatoare (“Family” sau “Group”), care identifica persoana dupa nume, o elimina din vector si decrementeaza “idx” – numarul persoanelor din familia/grupul respectiv.