Система за управление на БД със самолети за авиобаза

Изготвено от Таня Желева

Проектът представлява умалена версия на база данни за самолети. Поддържа се основна реализация на *REPL (read-evaluate-print-loop)* със сложности *O(n)*. Предвидена е и *Optimize* команда, която намаля сложността до *O(log(n))*. Информацията за всички самолети се съхранява в постоянната памет във вид на записи в текстов файл. Всеки самолетен запис има вида *Id Plane Type Flights*.

1 Архитектура

1.1 Plane

Класът `Plane` описва отделен запис в базата. Основните му характеристики са:

- Private:
  - `id` – Id на дадения самолет
  - `name` – име на самолет
  - `type` – типа на самолет
  - `flights` – брой извършени полети

- Public:
  - `Plane()` – конструктор по подразбиране

  - `Plane(int, const string&, const string&, int)` – конструктор, приема атрибути Id – уникално число, Plane – име, Type – тип и Flights – брой извършени полети. В себе си вика методите *`SetId(int), SetName(const string&), SetType(const string&), SetFlights(int).`*

  - `void SetId(int)` – сетър, задава стойност на *id* полето, като проверява дали подаденият параметър е валиден. В случай на невалидност хвърля *invalid_argument exception*.

  - `void SetName(const string&)` – сетър, задава стойност на полето *name*, като проверява дали подаденият параметър е валиден. В случай на невалидност хвърля *invalid_argument exception*.

- `void SetType(const string&)` – сетър, задава стойност на полето *type*, като проверява дали подаденият параметър е валиден. В случай на невалидност хвърля *invalid_argument exception*.

- `void SetFlights(int)` – сетър, задава стойност на полето *flights*, като проверява дали подаденият параметър е валиден. В случай на невалидност хвърля *invalid_argument exception*.

- `int GetId() const` – гетър, връща стойността на полето *id*.

- `const string GetName() const` – гетър, връща стойността на полето *name*.

- `const string GetType() const` – гетър, връща стойността на полето *type*.

- `int GetFlights() const` – гетър, връща стойността на полето *flights*.

- `bool operator<(const Plane& other)` – предефинира се оператора < за класа в зависимост от стойността на полето *id*.

- `friend ostream& operator<<(ostream& out, const Plane& plane)` – приятелска фукнция, предефинира се оператора << за класа *Plane*.

1.2 Node

Записът `Node` е основният елемент на класа *PlaneSearchTree*. Използва се рекурсивна структура със стойност (*value*) от тип Plane и указатели *left*, *right* отново от тип *Node**. Стойностите се инициализират чрез конструктор, който приема параметър от тип *const Plane&* за полето value. Left / right указателите се задават с nullptr. Допълнителни помощни методи са *`bool HasLeft() const`* и *`bool HasRight() const`*, които проверяват дали текущия елемент има ляв или десен наследник.

1.3 PlaneSearchTree

Класът представлява custom реализация на двоично дърво за търсене.

- Private:
  - `Node* root` – указател към корена на дървото.

  - `void DeleteNode(Node* node)` – премахва подадения връх, като рекурсивно изтрива наследниците му.

- `void InserNode(Node* node, const Plane& value)` – рекурсивно добавя нов връх към дървото със стойност *value* от тип *Plane*. Ако дървото няма върхове се добавя като корен. В противен случай се добавя като ляв наследник на листо, в зависимост от големината на *value*. По-големите стойности се добавят в дясно, по-малките в ляво.

- `Plane& FindNode(Node* node, int id)` – Рекурсивно проверява дали дървото съдържа *Plane* с *id* подадения параметър. Връща стойността *Plane* на намерения връх. Ако няма такъв връх, връща “празен самолет”.

- `bool ContainsNode(Node* node, int id)` – Рекурсивно проверява дали дървото съдържа Plane с *id* подадения параметър. Връща *true* или *false* в зависимост дали е намерен връх.

- `void InOrder(Node* node)` – обикаля дървото тип ляв наследник, корен, десен наследник.

➢ Public:

- `PlaneSearchTree()` – конструктор по подразбиране. Задава стойност *nullptr* на корена.

- `PlaneSearchTree(const PlaneSearchTree& other) = delete` – забранява *copy* конструктора за дадения клас.

- `PlaneSearchTree& operator=(const PlaneSearchTree& other) = delete` – забранява предефинирането на оператор = за дадения клас.

- `~PlaneSearchTree()` – деструктор, извиква в себе си метода *`Clear`*.

- `void Clear()` – изтрива тървото, като извиква в себе си метода *`DeleteNode`* с параметър корена.

- `void Reset()` – “рестартира” дървото, като задава стойност *nullptr* на корена.

- `Plane& Find(int id)` – търси в дървото връх със стойност *Plane* с *id* подадения параметър. Използва метода *`FindNode`* с параметри корена и подаденото *id*.

- `void Insert(const Plane& value)` – добавя нов връх към дървото. Използва метода *`InsertNode`* с параметри корена и подаденото *value*.

- `bool Contains(int id)` – проверява дали има връх със стойност *Plane* и *id* подадения параметър. Използва метода *`ContainsNode`* с параметри корена и подаденото *id*.

- `void Print()` – принтира стойностите на дървото. Използва метода *`InOrder`*, с параметър корена.

1.4 PlaneFactory

Класът `PlaneFactory` описва основните операции поддържани от базата.

- Private:
  - `PlaneSearchTree planesOptimized` – дърво от самолетите в базата. Използва се след изпълнение на *Optimize* командата.
  - `vector<Plane> planes` – списък от самолетите в базата.
  - `string fileName` – име на файла, който съдържа данните.
  - `bool isOptimizeUsed` – показва дали е използвана *Optimize* командата.
  - `void RemoveOptimized()` – премахва оптимизирането на базата, като занулява *planesOptimized* и *isOptimizedUsed*.
  - `int GetIdIndex(int id)` – връща индексът на подаденото *id* във вектора *planes*. Ако не намери индекс връща -1.
  - `void SearchOptimized(int id)` – изпълнява оптимизирана *Search* команда, като използва дървото *planesOptimized* вместо вектора *planes*.
  - `void CreateFilePlane(string record)` – по подаден параметър *record* създава обект от тип *Plane*. Необходимите стойности се вземат от стринга чрез регулярен израз.
  - `void ExtractFromFile()` – прочита данните за самолетите от текстовия файл.
- Public:
  - `PlaneFactory(const string& fileName)` – конструктор, задава стойности на *fileName* и *isOptimizedUsed*. Извлича данните от файла чрез метода *`ExtractFromFile`*.
  - `void Create(int id, const string& name, const string& type, int flights)` – създава нов самолет, като проверява дали даденото *id* съществува. Ако бъде намерено, хвърля *invalid_argument exception*. Премахва оптимизирането.

- `void Delete(int id)` – изтрива самолет, по подадено *id*. Ако такъв не бъде намерен хвърля *invalid_argument exception*.

- `void Update(int id, const string& attribute, const string& valueToString)` – обновява стойността на датен атрибут на самолет по подадено негово *id*. Ако такъв самолет не е намерен или е подаден невалиден атрибут хвърля *invalid_argument exception*.

- `void Show(int offset, int limit)` – принтира *limit* на брой самолети, като започва от *offset* разстояние от първия в списъка. При невалидни праметри или получена отризателна бройка за принтиране хвърля *invalid_argument exception.*

- `void Search(int id)` – търси самолет с подаденото *id*. Ако е изпълнена Optimize командата използва метода *`SearchOptimized`*. При невалидно *id* хвърля *invalid_argument exception*.

- `void Optimize()` – оптимизира базата, като добавя стойностите от вектора planes в дървото *planesOptimized*. Задава стойност *true* на *isOptimizedUsed* полето.

- `void Print() const` - принтира стойностите от *planes*.

- `void SaveToFile()` – запазва стойностите от *planes* във файла.

2. Четене на входните данни

Информацията за самолетите се съхранява в текстовия файл *PlanesDb.txt*. Потребителят въвежда команди. Обработват се чрез while цикъл, който съдържа в себе си try-catch блок. В случай на невалидни параметри, хвърлените изключения се прихващат, като се принтират съобщенията им. Има избор измежду следните команди:

- `create Id Plane Type Flights` – създаване на нов самолет. Илзползва се помощната функция *`Create(string& input, PlaneFactory& factory)`*. Чрез регулярен израз се “*match*”-ват отделните атрибути за създаването на самолет.

- `delete Id` – изтриване на самолет по негово *Id*. Използва се помощната функция *`Delete(string& input, int firstSpaceIndex, PlaneFactory& factory)`*. Чрез параметъра *firstSpaceIndex* се взема *Id* атрибута на самолета от *input* стринга.

- `update Id attribute attributeValue` – обновява атрибут на самолет с даденото *Id*. Използва се помощната функция *`Update(string& input, PlaneFactory&`*

factory). Чрез регулярен израз се “*match*”-ват отделните атрибути за обновяването на дадения самолет.

- `show offset limit` – принтира *limit* на брой записи от базата, като започва на разстояние offset от първия запис. Използва се помощната функция *Show(string& input, PlaneFactory& factory)*. Чрез регулярен израз се “*match*”-ват необходимите параметри от *input* стринга.

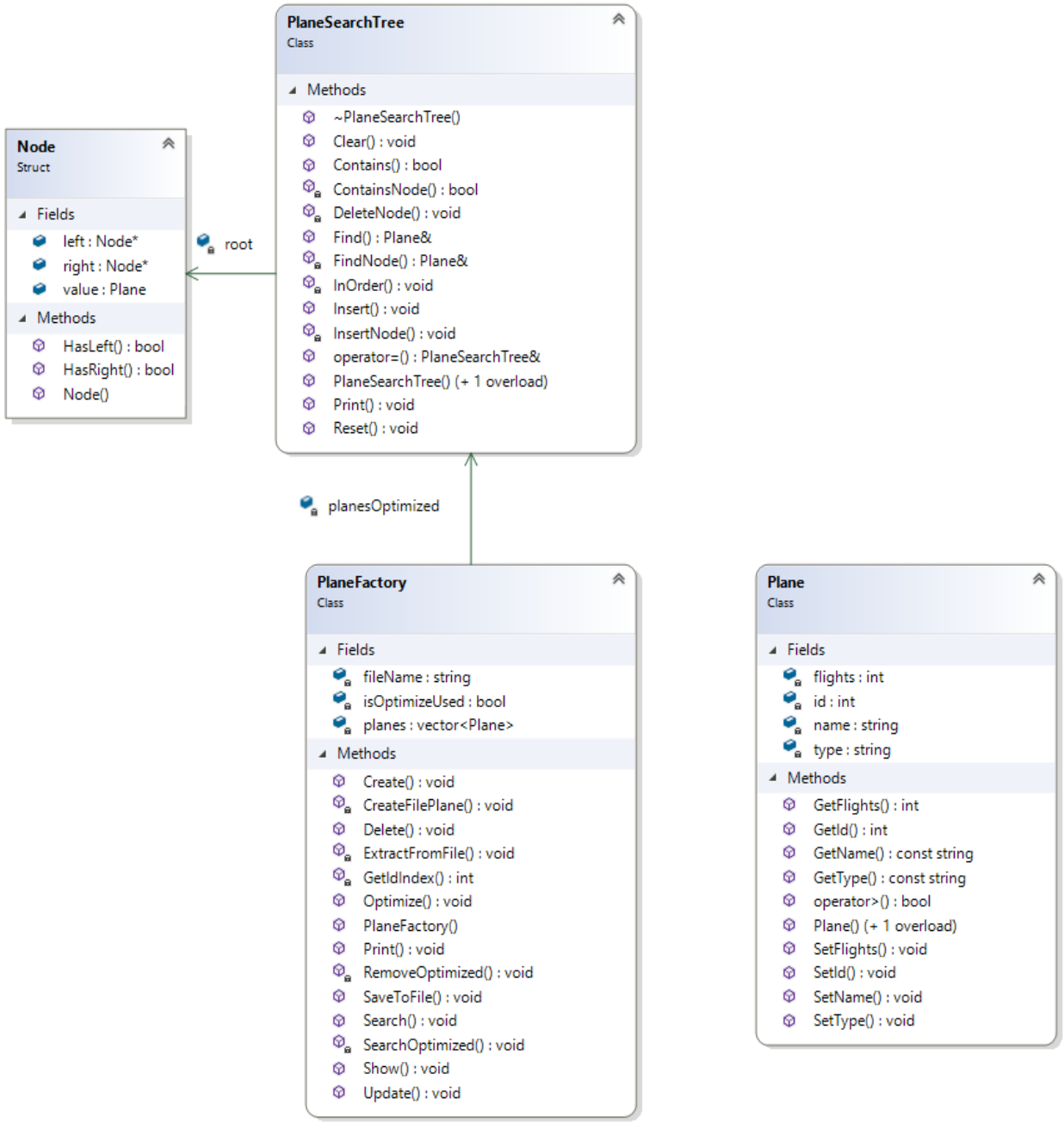
- `optimize` – оптимизира работата на базата. Сложността се намаля от *O(n)* до *O(n.log(n))*. В случай на добавяне / изтриване / обновяване на запис, базата се връща към предишното си състояние.

- `search Id` – принтира записа с даденото *Id*. Използва се помощната функция *Search(string& input, int firstSpaceIndex, PlaneFactory& factory)*. Чрез параметъра *firstSpaceIndex* се взема *Id* атрибута на самолета от *input* стринга.

- `print` – допълнителна команда, която принтира всички самолети от базата.

- `exit` – спира изпълнението на програмата.

3. Схема на проекта


PlaneSearchTree
Class
Methods
~PlaneSearchTree()
Clear() : void
Contains() : bool
ContainsNode() : bool
DeleteNode() : void
Find() : Plane&
FindNode() : Plane&
InOrder() : void
Insert() : void
InsertNode() : void
operator=() : PlaneSearchTree&
PlaneSearchTree() (+ 1 overload)
Print() : void
Reset() : void
Node
Struct
Fields
left : Node*
right : Node*
value : Plane
Methods
HasLeft() : bool
HasRight() : bool
Node()
root
planesOptimized
PlaneFactory
Class
Fields
fileName : string
isOptimizeUsed : bool
planes : vector<Plane>
Methods
Create() : void
CreateFilePlane() : void
Delete() : void
ExtractFromFile() : void
GetIdIndex() : int
Optimize() : void
PlaneFactory()
Print() : void
RemoveOptimized() : void
SaveToFile() : void
Search() : void
SearchOptimized() : void
Show() : void
Update() : void
Plane
Class
Fields
flights : int
id : int
name : string
type : string
Methods
GetFlights() : int
GetId() : int
GetName() : const string
GetType() : const string
operator>() : bool
Plane() (+ 1 overload)
SetFlights() : void
SetId() : void
SetName() : void
SetType() : void