Faculdade de Ciências e Tecnologia

Universidade Nova de Lisboa

FCT Boleia

Relatório de Algoritmos e Estruturas de Dados 2019/20 (MIEI)

Fase 3

Alunos : José Murta (55226) e Diogo Rodrigues (56153)

Curso: Mestrado Integrado em Engenharia Informática

Turno: P7 e P1

Docente prático: Sofia Cavaco

Boleia

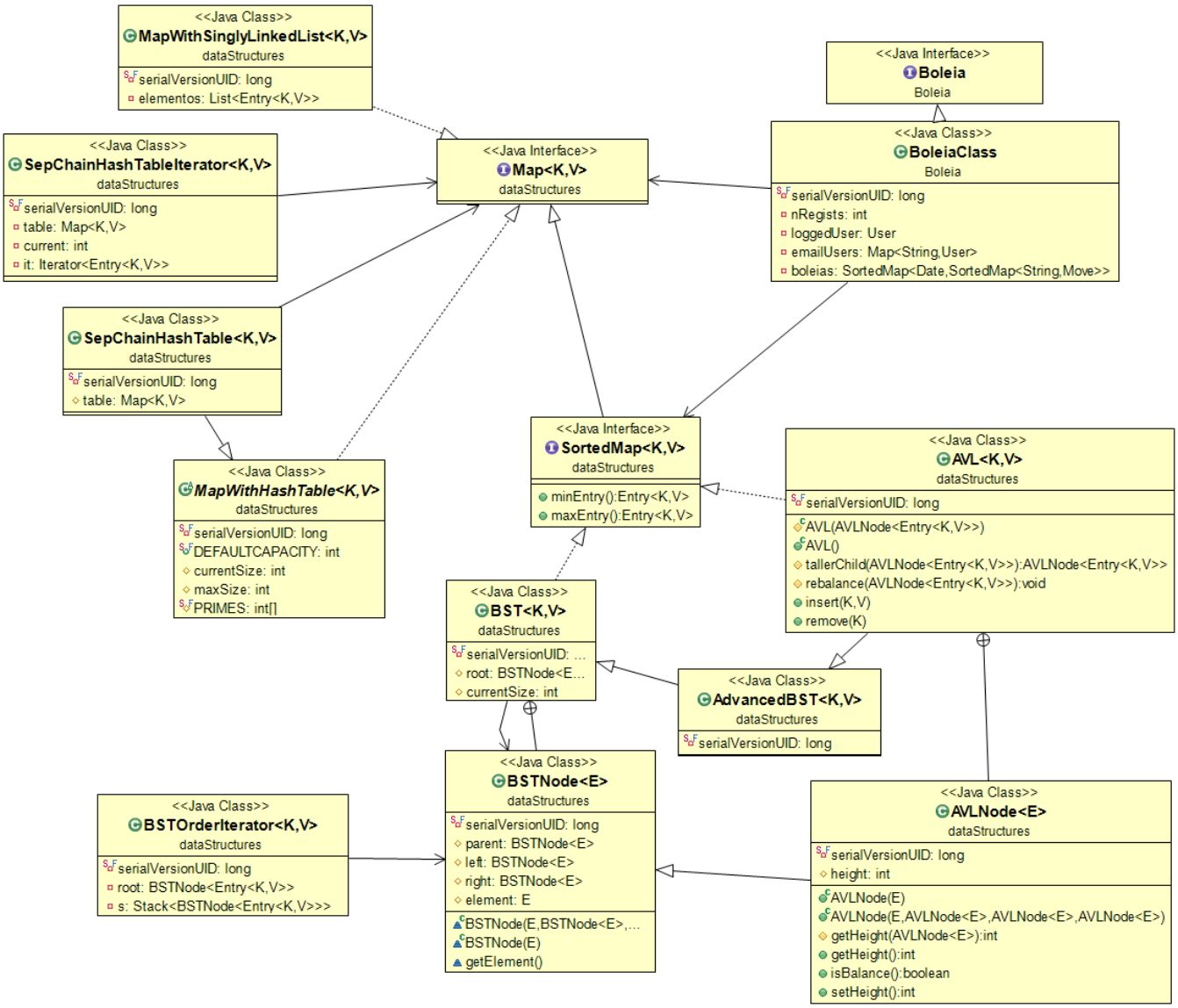


Figura 1 - Diagrama da classe Boleia

- ## Justificação das estruturas de dados utilizadas

Como estrutura de dados para implementação da TAD Map optámos por utilizar um Separate Chaining Hash Table (tabela de dispersão aberta) face a uma Linear Probing Hash Table (tabela de dispersão fechada) visto a primeira suportar a operação de remoção e as listas de colisão não se cruzarem. A sua complexidade temporal nas operações básicas (pesquisa, inserção e remoção) é constante (O(1)), no caso esperado, o que torna uma ED bastante viável no contexto do problema.

Na implementação do TAD Sorted Map escolhemos utilizar uma AVL Tree como estrutura de dados, pois visto ser uma árvore balanceada e organizada, impede a formação de cadeias e torna as operações de pesquisa, inserção, iteração e remoção bastante eficientes.

User

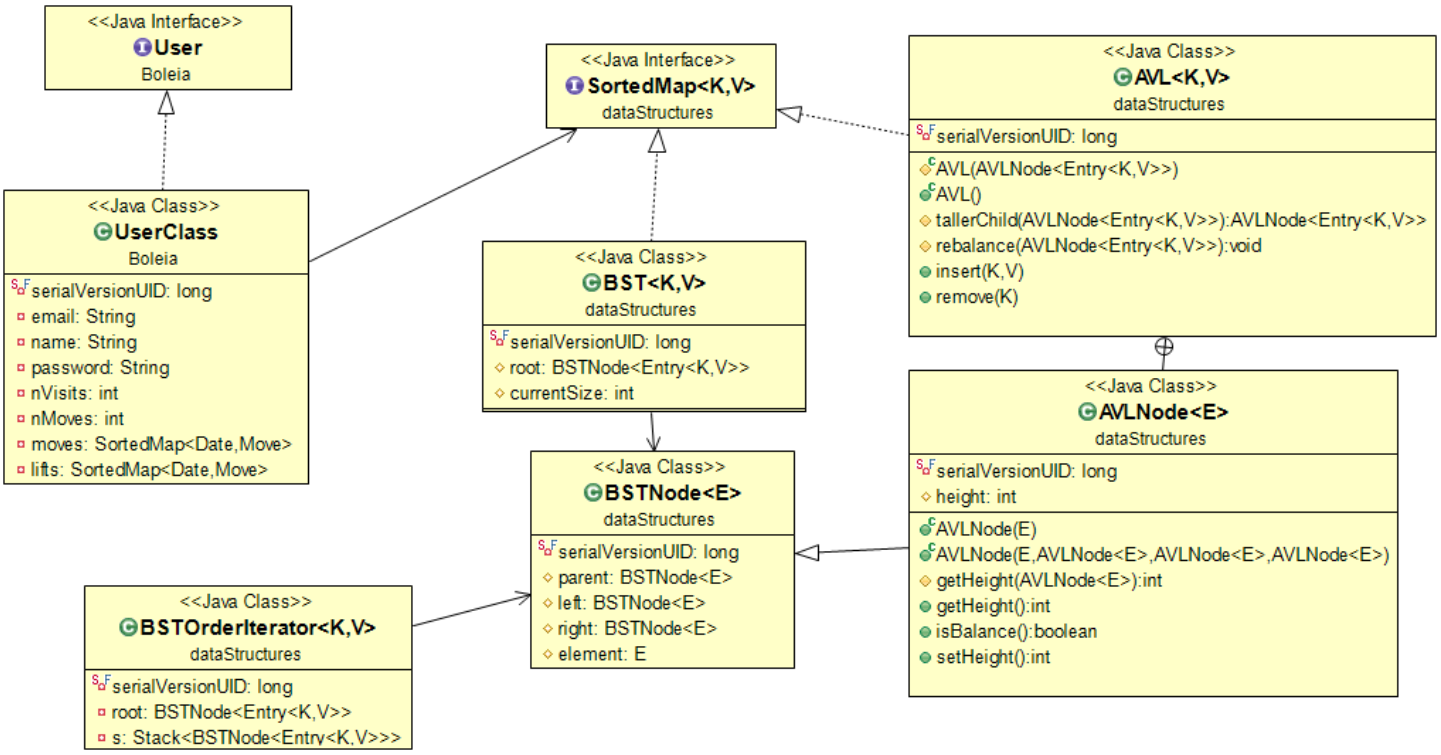


Figura 2 - Diagrama da Classe User

- ## Justificação das estruturas de dados utilizadas

Na classe UserClass para gerir as deslocações e as boleias do utilizador utilizámos o TAD Sorted Map, implementado com uma AVL Tree. Optámos por esta estrutura de dados pois tanto as boleias como as deslocações são inseridas e removidas constantemente e para tal necessitamos de uma ED eficiente nestas operações básicas. A complexidade temporal das operações básicas numa AVL Tree é, no caso médio e no pior caso, O(log(n)).

Move

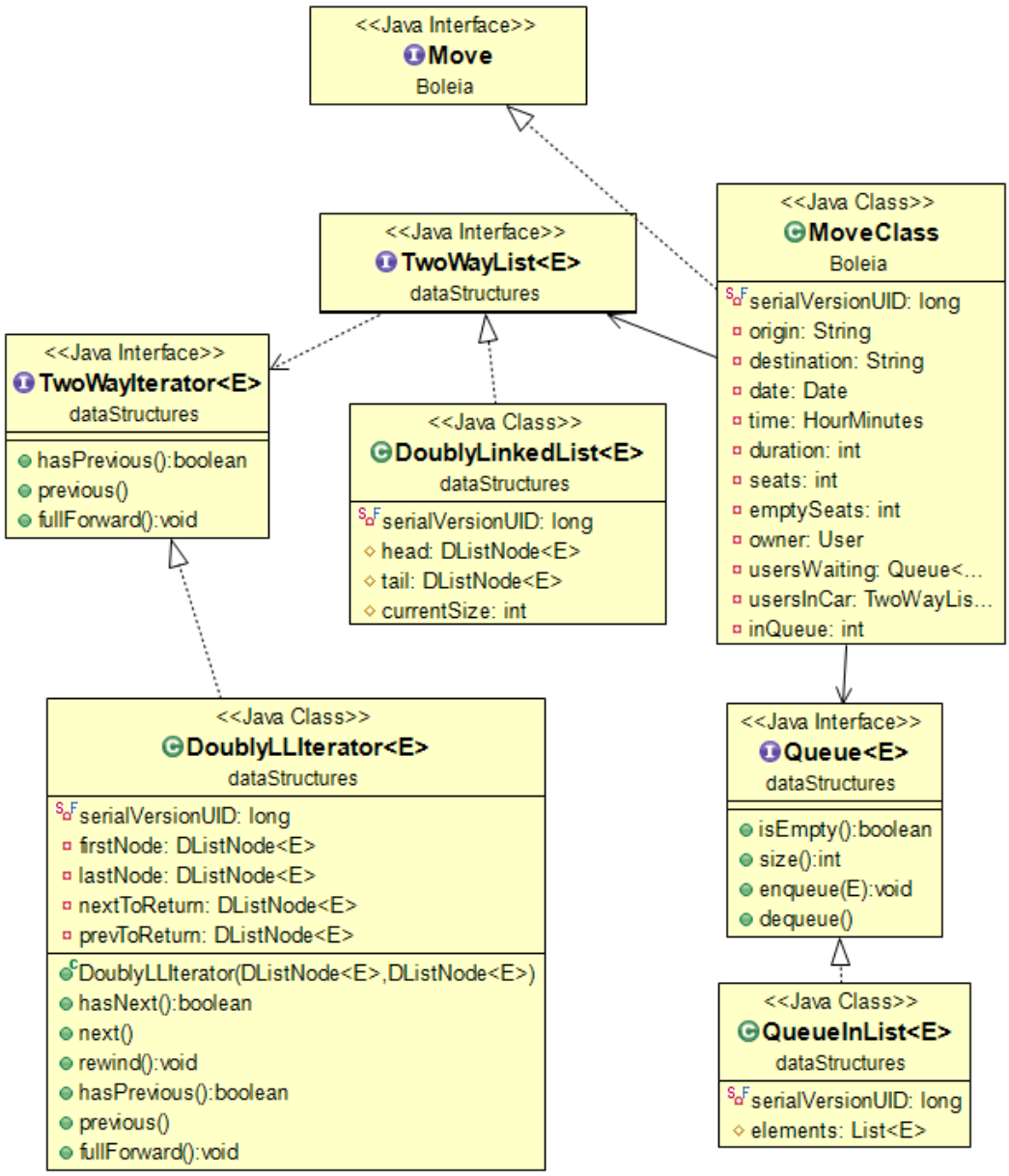


Figura 3 - Diagrama da Classe Move

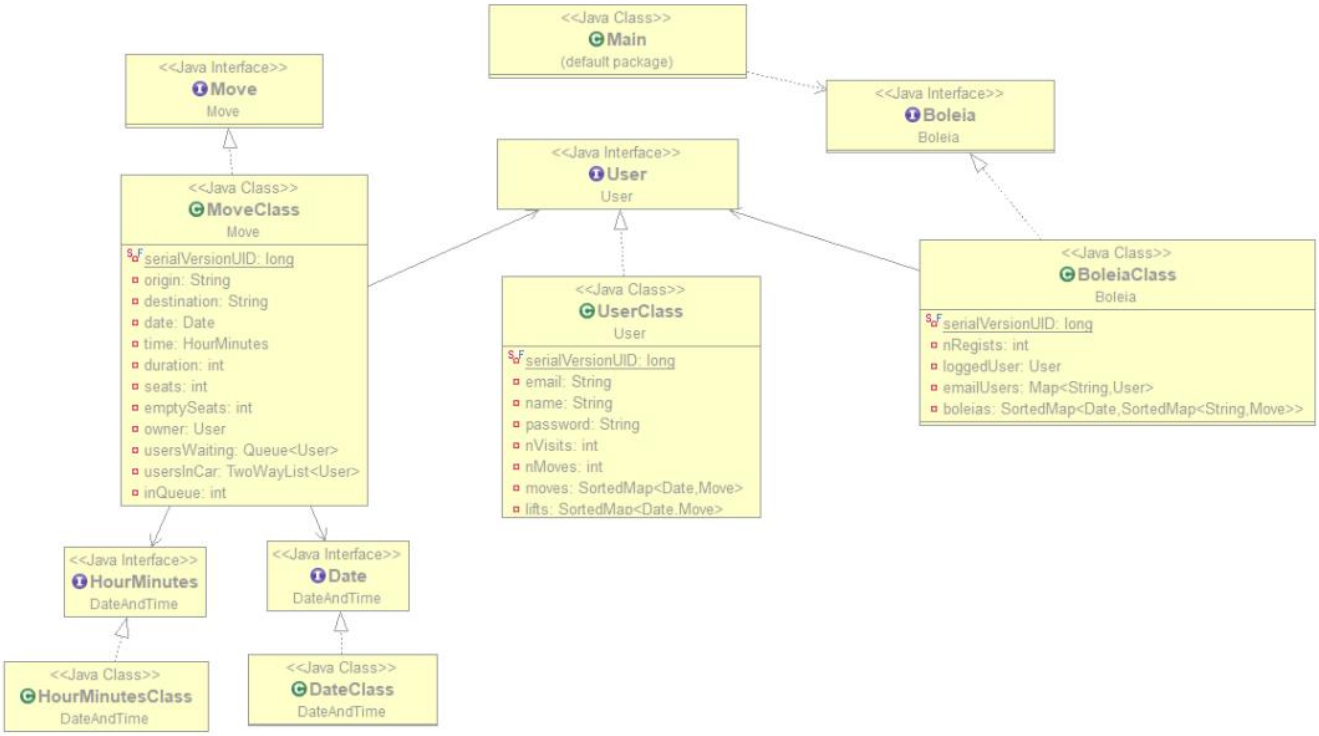
- ## Justificação das estruturas de dados utilizadas

Na Classe MoveClass decidimos utilizar a estrutura de dados QueueInList implementada com uma SinglyLinkedList para gerir os utilizadores em fila de espera para uma determinada deslocação, visto que a SinglyLinkedList é eficiente em inserções e remoções, sendo que ambos os métodos enqueue e dequeue, têm, em qualquer dos casos, complexidade temporal O(1).
Para os utilizadores que estão realmente adicionados a uma boleia, optámos por uma DoublyLinkedList visto ser recorrente nas operações básicas necessárias, cuja complexidade é O(1) em qualquer dos casos.

Descrição dos comandos do programa

- **Comando Regista:** Cumprindo todas as exceções, é criado um novo objeto do tipo User, e é adicionado o email desse utilizador ao SortedMap emailUsers, sendo a chave a String com o email do utilizador e o valor o próprio objeto User. Através de uma variável User (loggedUser) confirmamos o estado do programa (modo inicial ou sessão). A complexidade temporal deste método é: *BestCase:* O(1), *Averagecase:* O(1), *WorstCase:* O(n).
- **Comando Entrada:** Cumprindo todas as exceções, é realizada a entrada do utilizador caso a password seja compatível, sendo atualizada a variável loggedUser para este utilizador que entrou, e o estado do programa passa para modo sessão. A complexidade temporal deste método é: *BestCase:* O(1), *Averagecase:* O(1), *WorstCase:* O(n).
- **Comando Sai:** Caso o sistema esteja em modo sessão, o utilizador termina a sua sessão, tornando a variável loggedUser null, logo o programa fica em modo inicial. A complexidade temporal deste método é: *BestCase:* O(1), *Averagecase:* O(1), *WorstCase:* O(1).
- **Comando Termina:** Termina a execução do programa, caso o programa esteja em modo inicial (sem nenhum utilizador com sessão iniciada). A complexidade temporal deste método é: *BestCase:* O(1), *Averagecase:* O(1), *WorstCase:* O(1).
- **Comando Ajuda:** Imprime a lista de funcionalidades disponíveis consoante o estado do programa. A complexidade temporal deste método é: *BestCase:* O(1), *Averagecase:* O(1), *WorstCase:* O(1).
- **Comando Nova:** Cumprindo todas as exceções, é adicionada uma nova deslocação com determinadas características ao utilizador com sessão iniciada, adicionando esta deslocação ao SortedMap moves, sendo a chave a data da deslocação e o valor a própria deslocação, e ao SortedMap boleias da classe topo. A complexidade temporal deste método é: *BestCase:* O(1), *Averagecase:* O(log(n)), *WorstCase:* O(log(n)).
- **Comando Boleia:** Cumprindo todas as exceções enunciadas, é adicionada uma boleia ao utilizador com sessão enunciada, na deslocação criada por um dado utilizador numa dada data, adicionando o ao SortedMap lifts, a data da deslocação como chave e a própria deslocação como valor, e adicionando também o utilizador com sessão iniciada à TwoWayList usersInCar da própria deslocação. Caso essa deslocação esteja sem lugares vazios então o utilizador em sessão iniciada não será adicionado à TwoWayList usersInCar, mas sim à Queue usersWaiting. A complexidade temporal deste método é: *BestCase:* O(1), *Averagecase:* O(log(n)), *WorstCase:* O(log(n)).
- **Comando Retira:** Cumprindo todas as exceções, é removida a boleia, com uma data, ao utilizador com sessão iniciada. Remove do SortedMap lifts, da classe User, essa deslocação e remove o utilizador da TwoWayList usersInCar da própria deslocação, e caso estivesse algum utilizador na Queue usersWaiting, é retirado o primeiro utilizador desta fila e passado para usersInCar. A complexidade temporal deste método é: *BestCase:* O(1), *Averagecase:* O(log(n)), *WorstCase:* O(log(n)).

- **Comando Consulta:** Cumprindo todas as exceções enunciadas, são consultadas as informações da deslocação criada pelo utilizador com email dado, na data dada através da pesquisa Sortedmap boleias, em que a chave é a data, e o value é outro SortedMap em que a chave é o email do utilizador e o value é assim a deslocação que queremos consultar. A complexidade temporal deste método é: *BestCase:* O(1), *Averagecase:* O(log(n)), *WorstCase:* O(n).
- **Comando lista:** *Lista Minhas*: Lista todas as deslocações do utilizador que se encontra com sessão iniciada. Cumprindo todas as exceções,de forma a listar todas as deslocações iteramos o sortedMap moves no objeto User; *Lista Boleias*: Lista todas as boleias em que o utilizador com sessão iniciada se encontra inserido. Cumprindo todas as exceções,de forma a listar todas as boleias onde está inserido iteramos o sortedMap lifts no objeto User; *Lista DATA*:Lista todas as deslocações que existem no programa com a data que foi inserida pelo utilizador.Cumprindo todas as exceções, apenas são imprimidas deslocações com lugares vagos. De forma a listar todas as deslocações iteramos todas as deslocações existentes no sortedMap que é valor do SortedMap boleias na classe topo; *Lista EMAIL*: Lista todas as deslocações do utilizador com o email que foi dado. Cumprindo todas as exceções, iteramos o sortedMap de moves do utlizador com o email; *Lista Todas:* Lista todas as deslocações existentes no programa.Cumprindo todas as exceções, iteramos o sortedMap boleias que armazena todas as deslocações do programa. A complexidade temporal destes método é: *BestCase:* O(n), *Averagecase:* O(n), *WorstCase:* O(n), visto serem comandos de iteração.



Complexidade Espacial

- Nº esperado de Users = 10000 (derivado do enunciado);
- Nº esperado de Moves = M;
- Nº esperado de ocupantes de cada Move = NO;
- Nº esperado de Datas = D = M;
- Dimensão de cada User = DimU;
- Dimensão de cada Move = DimM;
- Dimensão de cada Data = DimD;
- Classes com Tabelas de Dispersão aberta (CDTA)= 1;
- Classes com AVLs Trees (CAVL) = 4;
- Nº de apontadores de cada nó de AVL (NA) = 4;
- Classes com Queues (CQ)=1;
- Nº esperado de Users em Queue = NUQ;
- Classes com DoublyLinkedLists (CDLL) = 1;
- Nº esperado de Users na DoublyLinkedList = NUDLL;
- Nº de apontadores de cada nó da DLL (NDLL) = 3;

Cálculo da complexidade espacial da aplicação:

(10000 x DimU) + (M x DimM) + (1.1 x 10000 + 2 x 10000) x CDTA + (M x NA x NO) x CAVL + (NUQ x CQ) + (NUDLL x NDLL x CDLL)