AEDA – Cartão Amigo Museus de Portugal

Trabalho 2 – Realizado por:
João de Jesus Costa - up201806560
João Lucas Silva Martins – up201806436
Tiago Duarte da Silva – up201806516

Descrição do Problema

Tema 4 – Cartão Museus de Portugal (Parte 2)

Complemente o sistema já implementado com as seguintes funcionalidades:

- De forma a direcionar as campanhas de *marketing*, mas também a melhorar os serviços prestados aos seus associados, a Direção-geral do Património Cultural pretende manter um registo dos museus visitados pelos detentores do cartão Amigo dos Museus de Portugal. Guarde numa **árvore binária de pesquisa** os museus visitados pelos aderentes à iniciativa, ordenados pelo número de aderentes que visitaram esse museu e, em caso de empate, por ordem alfabética. A identificação do Museu deve ainda incluir a localização (X,Y) do mesmo. Devem ser permitidas listagens várias, tirando partido da ordenação da árvore.
- Os museus e edifícios da Direção-geral do Património Cultural necessitam frequentemente de reparações ou manutenções, sendo as últimas normalmente planeadas. A Direção-geral do Património Cultural mantém um registo das várias empresas que podem efectuar as reparações e manutenções dos seus edifícios. O sistema deve ajudar na escolha da melhor empresa para efetuar este serviço. Para tal, as empresas são guardadas numa **fila de prioridade** ordenada pela sua número de serviços já efetuados para a Direção-geral do Património Cultural. A empresa que está no topo é a que tem mais serviços realizados. A empresa, para além do número de serviços realizados, é também caracterizada pela sua localização (X,Y). Para a reparação de determinado edifício, deve ser escolhida a empresa com mais experiência na realização deste tipo de serviços com a Direção-geral do Património Cultural e que se encontre a uma distância inferior a uma distância máxima a especificar.
- A Direcção-geral do Património Cultural mantém um registo de todos os seus trabalhadores (atuais ou antigos) numa **tabela de dispersão.** A manutenção do registo de trabalhadores antigos da empresa justifica-se porque, no caso de necessidade de contratação de um novo trabalhador, a entidade tem como preferência a contratação de trabalhadores já conhecidos. O trabalhador deve estar associado ao Museu em que trabalha (se é um trabalhador atual da empresa). Devem ser permitidas listagens ou pesquisas de trabalhadores, a especificar.

Novas classes/atributos

- Empresa de Reparações
- Trabalhador do Estado
- Novos atributos para Museus e a RPM

Problemas a resolver

- Guardar Museus mais visitados numa BST
- Guardar Empresas de Reparações numa priority queue com a Empresa com mais reparações no topo
- Escolher a melhor Empresa tendo em conta o número de reparações efetuadas e a sua distância a um Museu a reparar
- Guardar Trabalhadores do Estado (atuais e antigos) numa hashtable
- Relacionar cada trabalhador a um Museu (quando possível)
- Contratar um Trabalhador (com preferência de um Trabalhador antigo)
- Listagens tirando partido das vantagens de cada estrutura de dados

Solução para o problema

- Decisões relativas à organização do trabalho:
  - Tal como pedido, os Museus numa BST (std::set), as Empresas de Reparação numa priority queue (std::priority_queue) e os Trabalhadores do estado numa hashtable (std::unordered_set).
- Soluções práticas:
  - Para relacionar um Trabalhador com um Museu usámos o nome e as coordenadas do mesmo, pois estes são o que tornam cada Museu único.
  - Ao contratar um trabalhador, existe a opção de recontratar um trabalhador antigo (opção considerada prioritária), selecionando dos trabalhadores que não se encontram empregados de momento; ou criar um novo trabalhador e adicioná-lo à rede.
- Problemas na implementação da BST:
  - A ordem na BST é necessariamente a do enunciado (número de visitas, em caso de empate o nome), mas para saber se um Museu é igual a outro comparamos o seu nome e as suas coordenadas. Devido a isto, não podemos usar o find da STL para verificar a existência/encontrar um museu específico.
  - Para resolver esta questão, existem casos em que somos forçados a iterar sobre (no pior caso) toda a árvore (O(n)), comparando os Museus usando o operador == associado.

Estrutura de ficheiros

A estrutura dos ficheiros das classes antigas mantém-se (incluindo o ficheiro de configuração da RPM) com a adição das coordenadas dos Museus e dos novos ficheiros no config.

- O ficheiro dos Trabalhadores do Estado consiste num número identificando a quantidade de Trabalhadores a ler seguido pela informação deles.

- Cada Trabalhador guarda o seu nome, cc, contacto, *address*, data de nascimento, 1 se estiver empregue (0 caso contrário), seguido do nome e coordenadas do Museu que o emprega (caso esteja empregue).

```
2   // numero de state workers neste ficheiro
Joao Cucs                                                  // nome do primeiro worker
594909234                                                  // CC
916578392                                                  // contacto
Undefined Street :::::: 0000-000 :::::: rua dos Comboios   // endereço
-50202000                                                  // data de nascimento
1                                                          // esta contratado
Museu do Pao                                               // nome do museu que o contratou
1.00                                                       // coordenada X do museu que o contratou
2.00                                                       // coordenada Y do museu que o contratou
Carlos Pereira                                             // nome do segundo worker
174282932                                                  // CC
kaka.pereira@notgoogle.com                                 // contacto
rua das Fontainhas :::::: 4567-123 :::::: Carvalhos        // endereço
964310400                                                  // data de nascimento
0            // nao está contratado por isso não é seguido da informação do museu que o contratou
```

e.g. do conteúdo de um ficheiro de informação de Trabalhadores do Estado

Funcionalidades Implementadas

- As funcionalidades de CRUD foram mantidas (pois já estavam completas na totalidade) para as antigas classes (Museus, Empresas de Eventos, Cartões e Eventos) e foram completamente implementadas para Trabalhadores do Estado e Empresas de Reparações.

- Novas adições às interfaces:
  - Contratação e despedimento de Trabalhadores do Estado.
  - Reparar um Museu.
  - Efetuar a visita de um museu como detentor de um cartão *Amigo dos Museus de Portugal*

Funcionalidades Implementadas – Filtros

- As Empresas de Eventos e as Empresas de Reparações ambas derivam de uma classe base Empresa Base, visto partilharem bastante informação semelhante. A classe Empresa Base foi definida para este trabalho e a antiga classe Empresa de Eventos foi adaptada para derivar desta.

- Todas as operações de listagem e seleção de aglomerados de informação têm filtros distintos para ajudar a navegar pela grande quantidade de dados apresentados pela Rede. A acrescentar aos já presentes no trabalho anterior, destacam-se:

- **Filtro de Repair Enterprises:**
  - Por Endereço
  - Por Nome

- **Filtro de Trabalhadores do Estado**
  - Por Estado de Empregação
  - Por Nome

Funcionalidades Implementadas – Filtros e Sorts

Visto no trabalho anterior ter faltado foco nos sorts, agora todos os conjuntos de dados (excetuando os Museus por serem uma BST) têm um conjunto de propriedades pelas quais os podemos ordernar.

- **Sorts de Events:**
  - Por Preço
  - Por Nome
  - Por Ocupação
- **Sort de Enterprises**
  - Por Número de Eventos
  - Por Nome
- **Sort de Cartões:**
  - Por CC
  - Por nata de nascimento
  - Por Nome
- **Sorts de Repair Enterprises**
  - Por Nome
  - Por Número de Trabalhos Realizados
- **Sort de Trabalhadores do Estado**
  - Por Endereço
  - Por Nome

Funcionalidades a destacar

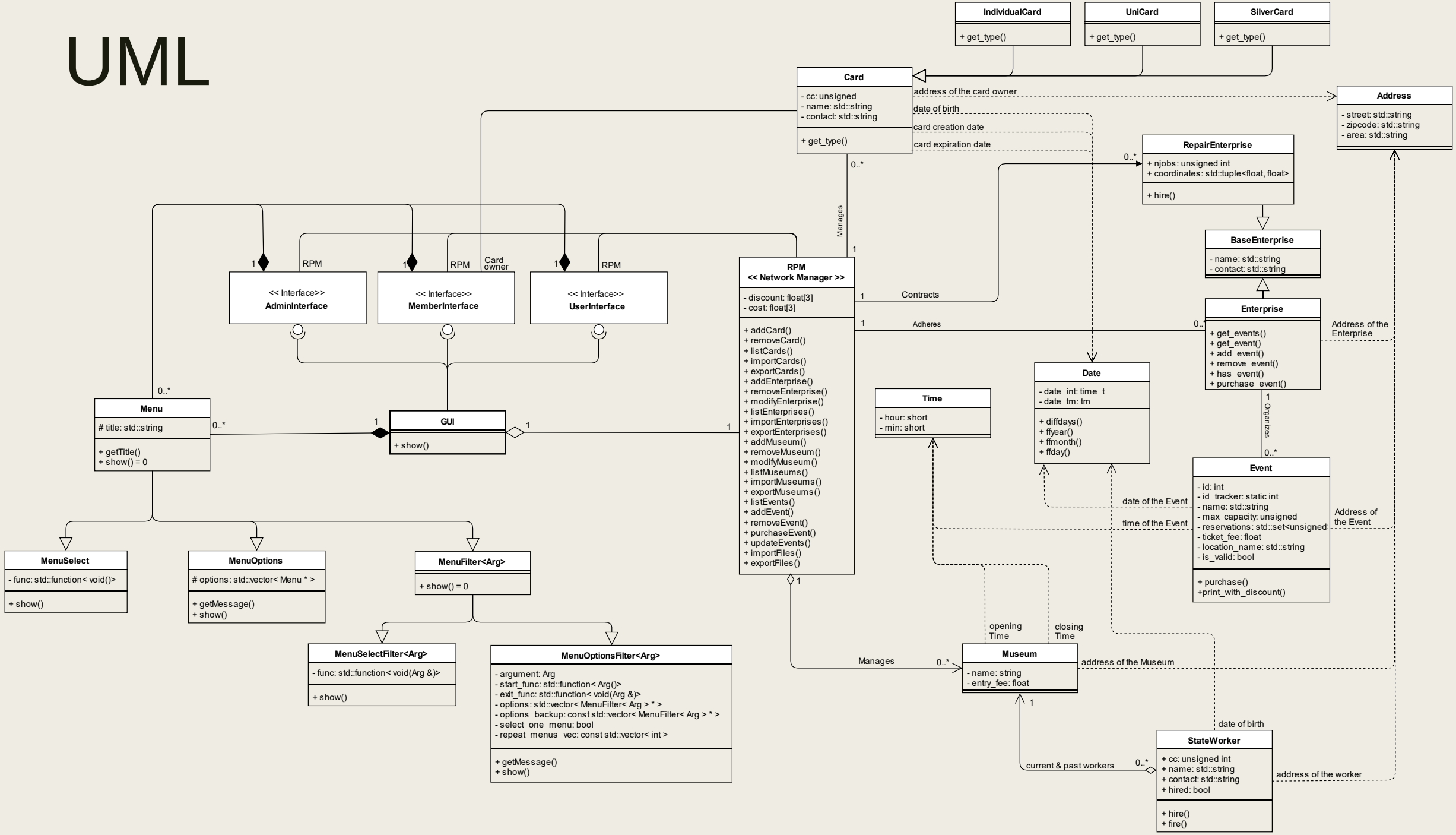
- A pesquisa de empresas (RepairEnterprises), para reparar um edifício (Museum) à escolha do utilizador:
  - Foi utilizada uma stack para guardar temporariamente os elementos que não correspondem à informação pretendida.
  - Após iterar sobre todos os elementos ou encontrar um que corresponda à pesquisa, os elementos na stack são reinseridos.

- O uso da classe auxiliar Menu, que facilitou:
  - A seleção de elementos da rede (quando era necessário visitar um museu, reparar um museu, etc …)
  - A remoção de vários elementos em simultâneo da rede
  - O uso de filtros e sorts na listagem de aglomerados

Principais dificuldades encontradas e esforço de cada elemento do grupo

- O trabalho foi distribuído de maneira homogénea por todos os elementos do grupo.
- As principais dificuldades encontradas foram:
  - O uso de estruturas não lineares dificultou o uso dos filtros dos nossos menus: os filtros deviam ter sido implementados com recurso a iteradores, por vez de vetores; a filtragem de ‘std::set’ continua eficiente, mas o código já não é tão geral.
  - A forma como o operador ‘<‘ foi definido para a **BST** de museus (a pedido do enunciado) dificulta a procura de museus nesta (comparação de igualdade entre museus não pode ser feita com o uso do operador ‘<‘ descrito). O grupo conclui que este operador não permite usufruir de todas as vantagens de uma **BST** (nomeadamente a procura).

UML


IndividualCard
+ get_type()
UniCard
+ get_type()
SilverCard
+ get_type()
Card
- cc: unsigned
- name: std::string
- contact: std::string
+ get_type()
address of the card owner
date of birth
card creation date
card expiration date
Address
- street: std::string
- zipcode: std::string
- area: std::string
RepairEnterprise
+ njobs: unsigned int
+ coordinates: std::tuple<float, float>
+ hire()
0..*
Manages
1
BaseEnterprise
- name: std::string
- contact: std::string
Enterprise
+ get_events()
+ get_event()
+ add_event()
+ remove_event()
+ has_event()
+ purchase_event()
Address of the Enterprise
1
Organizes
0..*
RPM
<< Network Manager >>
- discount: float[3]
- cost: float[3]
+ addCard()
+ removeCard()
+ listCards()
+ importCards()
+ exportCards()
+ addEnterprise()
+ removeEnterprise()
+ modifyEnterprise()
+ listEnterprises()
+ importEnterprises()
+ exportEnterprises()
+ addMuseum()
+ removeMuseum()
+ modifyMuseum()
+ listMuseums()
+ importMuseums()
+ exportMuseums()
+ listEvents()
+ addEvent()
+ removeEvent()
+ purchaseEvent()
+ updateEvents()
+ importFiles()
+ exportFiles()
Contracts
Adheres
RPM
RPM
Card owner
RPM
<< Interface>>
AdminInterface
<< Interface>>
MemberInterface
<< Interface>>
UserInterface
GUI
+ show()
Menu
title: std::string
+ getTitle()
+ show() = 0
0..*
Time
- hour: short
- min: short
Date
- date_int: time_t
- date_tm: tm
+ diffdays()
+ ffyear()
+ ffmonth()
+ ffday()
Event
- id: int
- id_tracker: static int
- name: std::string
- max_capacity: unsigned
- reservations: std::set<unsigned
- ticket_fee: float
- location_name: std::string
- is_valid: bool
+ purchase()
+print_with_discount()
date of the Event
time of the Event
Address of the Event
opening Time
closing Time
Museum
- name: string
- entry_fee: float
address of the Museum
Manages
StateWorker
+ cc: unsigned int
+ name: std::string
+ contact: std::string
+ hired: bool
+ hire()
+ fire()
date of birth
current & past workers
address of the worker
MenuSelect
- func: std::function< void()>
+ show()
MenuOptions
options: std::vector< Menu * >
+ getMessage()
+ show()
MenuFilter<Arg>
+ show() = 0
MenuSelectFilter<Arg>
- func: std::function< void(Arg &)>
+ show()
MenuOptionsFilter<Arg>
- argument: Arg
- start_func: std::function< Arg()>
- exit_func: std::function< void(Arg &)>
- options: std::vector< MenuFilter< Arg > * >
- options_backup: const std::vector< MenuFilter< Arg > * >
- select_one_menu: bool
- repeat_menus_vec: const std::vector< int >
+ getMessage()
+ show()