Projeto 6

Gestão de Contactos

Unidade Curricular de
Laboratório de Programação

2021/2022

Objetivos

- Manipulação de Mapas e Tabelas de Dispersão (*HashTables*);
- Uso da classe genérica `HashMap<K,V>` para implementar tabelas de dispersão.

Antes de Começar

- Faça *download* do ficheiro `studentsProject6.zip` acessível na página de LabP;
  Este ficheiro contém:
  - Uma classe `RunProject6.java` com a qual pode testar o código que vai implementar;
  - Ficheiros de *input* para a classe `RunProject6.java`;
  - Um ficheiro `expectedOutput.txt` que pode utilizar para comparar os resultados obtidos pela sua solução com os resultados esperados;
  - Ficheiros `ContactTest.java` e `ContactManagerTest.java` com testes JUnit para as classes que terá que implementar.
- No Eclipse,
  - crie um projeto Java e "arraste" para a pasta `src` os ficheiros `java` contidos em `studentsProject6.zip`;
  - arraste os ficheiros de *input* para a raiz do projeto.

De modo a melhor realizar este projeto, sugere-se que consulte a seguinte informação:

- Slides de AED sobre Mapas;
- Exercícios de AED sobre Mapas e Tabelas de Dispersão;
- Documentação da interface `Map` e da classe `HashMap` da biblioteca `java.util`.

Enunciado

Todos estamos familiarizados com a gestão de contactos que os nossos telemóveis permitem fazer. Podemos criar/visualizar/editar contactos que podem conter várias informações, tais como, por exemplo:

- Nome da pessoa
- Um ou mais números de telefone
- Morada
- Email
- Data do aniversário

A gestão desses contactos permite também que:

- Quando pesquisamos pelo nome do contacto, conseguimos aceder à informação do contacto de forma eficiente/direta;

- Quando recebemos um telefonema desse contacto, também há um acesso eficiente/direto à informação desse contacto, de tal modo que visualizamos no nosso telemóvel, de forma imediata, o nome de quem nos está a telefonar.

Neste projeto pretende-se definir uma classe `Contact.java` que nos permita representar contactos, e uma classe `ContactManager.java` que permita simular os comportamentos acima referidos no uso que os nossos telemóveis fazem desses contactos, usando para isso tabelas de dispersão.

A classe `ContactManager.java` deverá manter duas tabelas de dispersão, implementadas com recurso à classe `HasMap<K,V>`:

- Uma tabela de dispersão que usa como chave o nome de um contacto e como valor o contacto em causa, adiante referida como sendo a tabela de nomes-contactos.
- Uma tabela de dispersão que usa como chave o telefone (ou os telefones, no caso de existirem vários) de um contacto e como valor o contacto em causa, adiante referida como sendo a tabela de telefones-contactos.

Note que, para estas duas tabelas de dispersão, os valores são da classe `Contact.java` e que os objetos/instâncias dessa classe são partilhados por ambas as tabelas.

Os contactos existentes não devem estar duplicados, isto é, para cada nome, há um único contacto com esse nome e, para cada telefone, há um único contacto com esse telefone.

Quando se tenta editar um contacto, caso já exista anteriormente um contacto com o mesmo nome ou telefone, esse contacto anterior é alterado em conformidade com o pedido, **NÃO** é criado um novo contacto.
Isso pode implicar remover/inserir/alterar pares chave-valor de uma ou das duas tabelas de dispersão mantidas pelo `ContactManager`, de modo a mantê-las coerentes entre si.

A classe `ContactManager` deve ainda registar internamente, sob a forma de texto, informação sobre quais os contactos que foram criados (deve manter um "*log*" das criações de contactos).

A classe `Contact`

Os objetos da classe `Contact` podem ter o nome do contacto, um ou mais telefones, uma morada, um email e uma data do aniversário.
Só é obrigatório que um contacto tenha o nome OU um telefone associado (poderá ter só o nome, sem nenhum telefone, ou um telefone, sem um nome associado, ou ter ambos), todas as restantes informações podem estar omissas.

Os objetos da classe `Contact` devem ter pelo menos um construtor público, *getters* e *setters* para os atributos, e implementar o método `toString` de forma adequada. Em particular, esta classe deve ainda ter um método `copy`, que retorna uma cópia profunda do contacto, e um método `upDatePhone`, que permite alterar um telefone conhecido do contacto. O método `getPhones` deve retornar toda a lista de telefones do contacto e o método `getPhone` deve retornar o primeiro dessa lista (veja nas classes dadas `ContactTest`, `ContactManagerTest` e `RunProject6` quais as assinaturas requeridas para os métodos desta classe).

A classe `ContactManager`

A classe `ContactManager` deve ter pelo menos o construtor e os métodos públicos seguintes:

- `ContactManager(String managerName)` que recebe o nome da pessoa que é "dona" dos contactos e cria tabelas de contactos vazias para esse *manager*.
- `Contact contactByName(String contactName)` que cria ou localiza um contacto dado o seu nome. Recebe o nome de um contacto e, caso exista um contacto com esse nome na tabela de dispersão de nomes-contactos, localiza-o e retorna-o, caso contrário cria um novo, insere-o nessa tabela e devolve o contacto criado.
  Caso este método tenha criado um novo contacto, deve ainda registar no *log* uma mensagem com a seguinte forma:
  `New contact created: contacName`
- `Contact contactByPhone(String phoneNumber)` que cria ou localiza um contacto dado um número de telefone. Recebe um número de telefone e, caso exista um contacto com esse telefone na tabela de telefones-contactos, localiza-o e retorna-o, caso contrário, cria um novo, insere-o nessa tabela e devolve o contacto criado.
  Caso este método tenha criado um novo contacto, deve ainda registar no *log* uma mensagem com a seguinte forma:
  `New contact created: phoneNumber`
- `boolean editContactName(String contactKey, String newName)` que recebe uma chave (que poderá ser um nome ou um telefone) de um contacto já existente e um nome e permite editar o nome desse contacto (isto é, inserir um nome nesse contacto ou alterar o já existente).
  Note que este método deve retornar `false` se não existir nenhum contacto com essa chave ou já existir um contacto na tabela de dispersão de nomes-contactos com o mesmo nome `newName`. Se isso não acontecer, deve retornar `true`, e há ainda duas situações possíveis:
  - Caso o contacto que está a ser editado não esteja ainda presente na tabela de dispersão de nomes-contactos (não era conhecido um nome para esse contacto), então a essa tabela terá que ser associado um novo par chave-valor.
  - Caso esse contacto já existisse na tabela de nomes-contactos, com um nome diferente do dado, essa tabela terá que sofrer uma alteração.

- `boolean editContactPhone(String contactKey, String newPhone, String oldPhone)` que recebe uma chave de um contacto já existente e um telefone e permite editar um telefone desse contacto (isto é, inserir um novo telefone ou alterar um já existente). Se se trata de alterar um telefone já associado a esse contacto, `oldPhone` indica qual o telefone desse contacto a ser alterado, caso contrário `oldPhone` deverá ser `null`. Pode assumir que `phone` e `oldPhone` são `Strings` que só contêm algarismos.
  Note que este método deve retornar `false` se não existir nenhum contacto com essa chave ou já existir um contacto na tabela de dispersão de nomes-contactos com o mesmo telefone `newPhone`. Se isso não acontecer, deve retornar `true`, e há ainda duas situações possíveis:
  - Caso `oldphone` seja `null, newPhone` é um telefone novo para o contacto que está a ser editado (o qual não está ainda presente na tabela de dispersão de telefones-contactos) e a essa tabela terá que ser associado um novo par chave-valor.
  - Caso `oldphone` não seja `null, newPhone` é uma alteração de um telefone para o contacto que está a ser editado, pelo que já deve existir um par com chave `oldPhone` na tabela de telefones-contactos e então essa tabela terá que sofrer uma alteração.
- `boolean editContactOptionalInfo(String contactKey, String address, String email, LocalDate date)` que recebe uma chave dum contacto já existente e valores para informação (opcional) a adicionar a esse contacto. Pode assumir que pelo menos uma dessas informações, indicadas pelos parâmetros restantes, terá que ser diferente de null.
  Uma chamada deste método altera ou adiciona ao contacto com a chave dada a informação passada num dos restantes argumentos de chamada do método, se o argumento em causa for diferente de `null`.
  Este método deve retornar false se não existe um contacto com essa chave.
- `String getNameFromPhone(String phone)` que recebe um número de telefone e retorna o nome do contacto que tem esse telefone. Deve retornar `null` se esse número de telefone não existe (na tabela de telefones-contactos) ou se não tem nenhum nome associado.
- `String getPhoneFromName(String name)` que recebe um nome de um contacto e retorna o primeiro telefone desse contacto. Deve retornar `null` se esse nome não existe (na tabela de nomes-contactos) ou se não tem nenhum telefone associado.
- `String getInfo(String contactKey)` que recebe uma chave de um contacto e retorna a representação textual de toda a informação associada a esse contacto. Este método deve retornar `null` se não existe nenhum contacto com essa chave.
- `void remove (String contactKey)` que recebe uma chave de um contacto e remove todas as ocorrências do contacto que tem essa chave das duas tabelas de dispersão existentes.

O que entregar

Deve criar o ficheiro `P6fcxxxxx.zip`, onde `xxxxx` é o seu número de aluno, contendo os ficheiros `Contact.java` e `ContactManager.java`