Vectores de objectos

Mestrado Integrado em Engenharia Informática FCT UNL
http://ctp.di.fct.unl.pt/miei/ip/

Corpo Docente 2020/2021
António Ravara, Artur Miguel Dias, Bernardo Toninho,
Ema Vieira, Inês Fernandes, Margarida Mamede,
Miguel Monteiro, Rui Nóbrega

Agenda de Contactos

- **Objectivo**
  - Manipular uma agenda de contactos.
- **Descrição e Funcionalidades**
  - Cada contacto na agenda caracteriza-se por um nome, um telefone e um *e-mail*. Na agenda, o contacto é identificado pelo seu nome.
  - Deve ser sempre possível consultar e/ou alterar o telefone e o email de um dado contacto, indicando o nome do contacto.
  - Pode-se registar novos contactos e remover contactos existentes.
  - É sempre possível listar a informação de todos os contactos.
- **Interacção com o utilizador**
  - A interface de utilização do programa é feita através de comandos (interpretador de comandos).

Interpretador de comandos

- O programa principal deve dar ao utilizador a possibilidade de execução de diversas operações numa agenda, no número que o utilizador pretender.
- Interface de utilização do programa: comandos

```
> AC (adiciona um contacto)
> RC (remove um contacto)
> GP (consulta o telefone de um contacto)
> GE (consulta o e-mail de um contacto)
> SP (actualiza o telefone de um dado contacto)
> SE (actualiza o e-mail de um dado contacto)
> LC (lista todos os contactos existentes na agenda)
> Q  (sair)
```

Definição dos comandos

- **Adicionar novo contacto:**
  - Adiciona o novo contacto à agenda, se não existir um contacto com o mesmo nome.

```
> AC Joana@fct.unl.pt 99999999 Joana Dias
Contact added.
> AC Joana@fct.unl.pt 99999999 Joana Dias
Contact already exists.
> AC Joana@gmail.com 91999999 Joana Horas
Contact added.
```

Definição dos comandos

- **Remover um contacto:**
  - Remove o contacto da agenda, se existir um contacto com o nome dado.

```
> RC Joana Dias
Contact removed.
> RC Joana Dias
Cannot remove contact.
> AC Joana@fct.unl.pt 99999999 Joana Dias
Contact added.
```

Definição dos comandos

- **Consultar telefone:**
  - Consulta o telefone do contacto, se existir um contacto com o nome dado.

```
> GP Joana Dias
99999999
> GP Joana Meses
Contact does not exist.
```

Definição dos comandos

- **Consultar *e-mail*:**
  - Consulta o *e-mail* do contacto, se existir um contacto com o nome dado.

```
> GE Joana Dias
Joana@fct.unl.pt
> GE Joana Meses
Contact does not exist.
```

Definição dos comandos

- **Actualiza o telefone:**
  - Actualiza o telefone do contacto, se existir um contacto com o nome dado.

```
> SP 253253253 Joana Dias
Contact updated.
> SP 253253253 Joana Meses
Contact does not exist.
> GP Joana Dias
253253253
```

Definição dos comandos

- **Actualiza o *e-mail*:**
  - Actualiza o *e-mail* do contacto, se existir um contacto com o nome dado.

```
> SE JoanaEu@tu.ele.pt Joana Dias
Contact updated.
> SE JoanaEu@tu.ele.pt Joana Meses
Contact does not exist.
> GE Joana Dias
JoanaEu@tu.ele.pt
```

Definição dos comandos

- **Listagem de contactos:**
  - Lista os contactos.

```
> LC
Joana Dias;JoanaEu@tu.ele.pt;253253253
Joana Horas;Joana@gmail.com;91999999
```

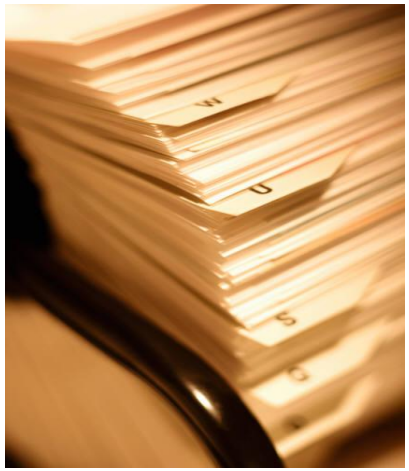
  - Lista os contactos – o que acontece se a lista estiver vazia?

```
> LC
Contact book empty.
```

Agenda de Contactos

- Desenvolver em Java uma classe `ContactBook` cujos objectos são agendas de contactos.
  - Cada objecto desta classe contém um conjunto de contactos.

    Logo é necessário definir a classe `Contact`, cujos objectos são contactos.



Estrutura da Aplicação

Interface com o utilizador

```
public class Main { ...
  public static void main(...) {
    ContactBook cBook;
    ...
  }
  private static void addContact(...){...}
  private static void importContacts(...){...}
  private static void deleteContact(...){...}
  private static String getPhone(...){...}
  private static String getEmail(...){...}
  private static void setPhone(...){...}
  private static void setEmail(...){...}
  private static void listAllContacts (...){...}
  ...
}
```

Classes do domínio

```
public class ContactBook {
  public boolean hasContact(...){...}
  public int getNumberOfContacts(){...}
  public void addContact(...){...}
  public void importContacts(...){...}
  public String getPhone(...){...}
  public String getEmail(...){...}
  public void deleteContact(...){...}
  public void setPhone(...){...}
  public void setEmail(...){...}
  public ContactIterator iterator() {...}
}
```

```
public class ContactIterator {
  public boolean hasNext(){...}
  public Contact next(){...}
}
```

```
public class Contact {
  public Contact(...){...}
  public String getName() {...}
  public String getPhone() {...}
  public String getEmail() {...}
  public void setPhone(String phone) {...}
  public void setEmail(String email) {...}
  public boolean equals(Contact other) {...}
}
```

Contacto

- Cada contacto caracteriza-se por um **nome**, um **telefone** e um **email**.
- Para criar objectos da classe `Contact` são necessários três argumentos, todos do tipo `String`: **email, telefone e nome do contacto**.
- Deve existir uma funcionalidade que indica se dois contactos são iguais, isto é se têm o mesmo nome.

Interface da classe Contact

- Operações reconhecidas (interface de `Contact`):

`String getName()`
Devolve o nome do contacto

`String getPhone()`
Devolve o telefone do contacto

`String getEmail()`
Devolve o e-mail do contacto

`void setPhone(String phone)`
Altera o telefone do contacto para phone
pre: `phone!= null`

`void setEmail(String email)`
Altera o email do contacto para email
pre: `email != null`

`boolean equals(Contact otherContact)`
Retorna `true`, caso o nome do contacto corrente seja igual ao nome do contacto dado
pre: `otherContact != null`

Programando a classe `Contact`

```
public class Contact {
    private String name;
    private String phone;
    private String email;

    // pre: name != null && phone != null && email != null
    public Contact(String name, String phone, String email) {
        this.name = name;
        this.phone= phone;
        this.email= email;
    }
    public String getName() { … }
    public String getPhone() { … }
    public String getEmail() { … }
    public void setPhone(String phone) { … }
    public void setEmail(String email) { … }
    public boolean equals(Contact otherContact) { … }
}
```

Programando a classe `Contact`

```
public class Contact {
    private String name;
    private String phone;
    private String email;

    // pre: name != null && phone != null && email != null
    public Contact(String name, String phone, String email) {…}
    public String getName() { … }
    public String getPhone() { … }
    public String getEmail() { … }
    // pre: phone != null
    public void setPhone(String phone) { … }
    // pre: email != null
    public void setEmail(String email) { … }
    // pre: otherContact != null
    public boolean equals(Contact otherContact) { … }
}
```

Nota: o conjunto de operações modificadoras suportadas não permite alterar o nome de um contacto

Programando a classe `Contact`

```
public class Contact {
    private String name;
    private String phone;
    private String email;

    public Contact(String name, String phone, String email) {…}

    public String getName() { … }
    public String getPhone() { … }
    public String getEmail() { … }
    public void setPhone(String phone) { … }
    public void setEmail(String email) { … }
    public boolean equals(Contact otherContact) { … }
}
```

Método equals

Devolve **true**, se o objecto corrente (i.e., `this`) é "equivalente" ao objecto do mesmo tipo passado como argumento; e **false**, caso contrário.

Programando a classe `Contact`

```
public class Contact {
    private String name;
    private String phone;
    private String email;

    public Contact(String name, String phone, String email) {…}

    public String getName() { … }
    public String getPhone() { … }
    public String getEmail() { … }
    public void setPhone(String phone) { … }
    public void setEmail(String email) { … }
    public boolean equals(Contact otherContact) { … }
}
```

O teste de equivalência envolve normalmente todas a variáveis de instância, mas podemos usar apenas algumas, se a especificação do problema o permitir.
Exemplo: podemos usar apenas o nome para comparar contactos, se assumirmos que o nome é um identificador único de contacto e que, portanto, dois contactos com nomes iguais se consideram iguais.

Comparar objectos: o método `equals`

Equivalência de objectos:

- Dada uma classe `Type`, a operação em `Type` para verificar a equivalência de 2 objectos `Type` será a seguinte:

```
public boolean equals(Type anotherObject)
```

Devolve **`true`**, se o objecto corrente (i.e., **`this`**) é equivalente ao objecto do mesmo tipo passado como argumento; e **`false`**, caso contrário.

- Em Java, os operadores `==` e `!=` comparam **referências** de objectos.
  - Ou seja, `a == b` testa se `a` e `b` são, de facto, **duas referências** para **o mesmo objecto em memória** , ou seja, **compara a identidade de objectos**
  - Em contrapartida `a != b` testa se `a` e `b` são duas referências para objectos distintos, ou seja, com diferentes identidades, mesmo que sejam equivalentes
  - Exemplo:

```
Contact a = new Contact("Joao Dias",219999,"jd@fct.unl.pt");
Contact b = new Contact("Joao Dias",210000,"jd@fct.unl.pt");
boolean x = (a == b); //false, a e b têm identidades diferentes!
boolean y = a.equals(b); // true, só estamos a comparar os nomes
```

Programando a classe `Contact`

```
public class Contact {
    private String name;
    private String phone;
    private String email;
    /** pre: name != null && email!= null */
    public Contact(String name, String phone, String email){…}
    public String getName() { … }
    public String getPhone() { … }
    public String getEmail() { … }
    /** pre: phone != null */
    public void setPhone(String phone) { … }
    /** pre: email != null */
    public void setEmail(String email) { … }
    /** pre: otherContact != null */
    public boolean equals(Contact otherContact) {
        return name.equals(otherContact.getName());
    }
}
```

Método `equals` da classe String, que verifica se duas sequências de caracteres são iguais.

Agenda de Contactos

- Uma agenda de contactos é um conjunto de vários contactos.
  - Cada contacto é identificado pelo seu nome.
  - Não podem existir contactos diferentes com o mesmo nome.