Programação Orientada Pelos Objectos

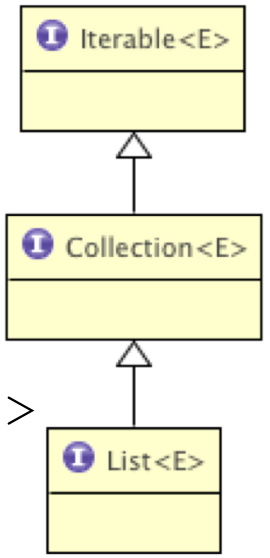
Colecções

Colecção

- Uma colecção é um grupo de elementos, sem indicações especiais sobre a existência ou não de uma ordem entre si, ou até se existem elementos repetidos ou não

Lista

- Uma lista, ou sequência, é uma colecção de elementos com uma ordem, podendo por norma ter elementos repetidos
- É possível ter acesso a um elemento da lista indicando a respectiva posição na lista, pesquisar elementos ou ainda inserir um elemento numa determinada posição
- Em Java




```
public interface List<E> extends Collection<E>
```

<E>

Interface `List<E>`

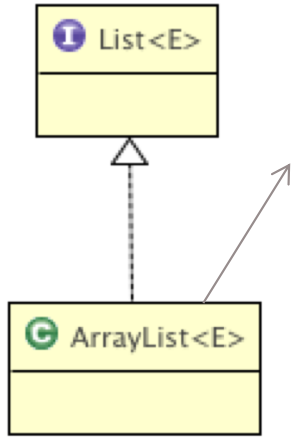
Métodos mais utilizados	
boolean add(E element)	Adiciona o elemento indicado no fim da lista
void add(int index, E element)	Insere na lista, na posição indicada o elemento
E get(int index)	Devolve o elemento que se encontra na lista na posição indicada
boolean isEmpty()	Verifica se a lista não tem elementos
E remove(int index)	Remove e devolve o elemento que se encontra na lista na posição indicada
E set(int index, E element)	Substitui o elemento que se encontra na lista na posição indicada pelo novo elemento
int size()	Devolve o número de elementos na lista

Tipo genérico em `List`

- Questão:
  - De que tipo podem ser os objectos inseridos em `List<E>` usando o método `add(E elem)`?
    - Tem de satisfazer o princípio da substituição
    - Tem de manter intactas todas as regras relativas a tipos estáticos e dinâmicos
- Resposta:
  - Objectos do tipo `E` e de qualquer subtipo de `E`

Classe `ArrayList<E>`

- **`ArrayList`** é uma implementação da interface **`List`**
- São disponibilizados os métodos definidos no interface List<E>
- A classe **`ArrayList`** é uma classe genérica
  - "colecciona" objectos do tipo **`E`**




```
public class ArrayList<E> extends AbstractList<E>
                          implements List<E>, ...
```

7

Iteradores para a lista

Voltando ao nosso exemplo

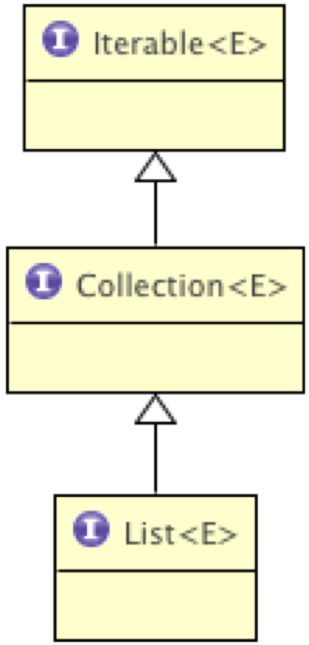
- Podemos utilizar **`List`** com implementação em **`ArrayList`** para guardar os nossos objectos
- Recorde-se que precisamos de dois tipos de iteradores
  - Iterador específico, que apenas devolve objectos com determinada característica
  - Iterador geral, que devolve todos os objectos da colecção
- Vamos verificar se existem iteradores da linguagem Java que possam ser úteis

Interface `Iterator<E>`

- Em Java, qualquer classe que implemente a interface **`Collection<E>`** tem de implementar a interface **`Iterable<E>`**
- Temos assim o iterador **`Iterator<E>`**

Métodos associados a `Iterable<T>`	
Iterator<T> iterator()	Devolve um iterador sobre os elementos da colecção

Métodos associados a `Iterator<E>`	
boolean hasNext()	Devolve **`true`** se a iteração tem mais elementos
E next()	Devolve o próximo elemento na iteração
void remove()	Remove da colecção o último elemento devolvido pelo iterador (operação opcional)



Iterador `for`(each)

- As iterações sobre colecções também podem ser feitas de uma forma compacta através do iterador *foreach*
  - "esconde" a criação de **`Iterator<E>`**, o teste do fim de iteração e o avanço para o próximo elemento
  - Significado: "*com cada elemento elem de tipo E obtido da colecção iterável, executa o bloco de instruções*"

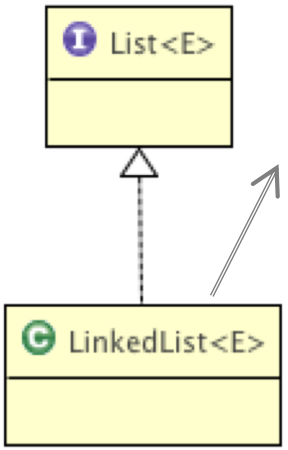
```
for (E elem : ColecçãoIterável<E>)
    bloco de instruções
```

Iterador `ListIterator<E>`

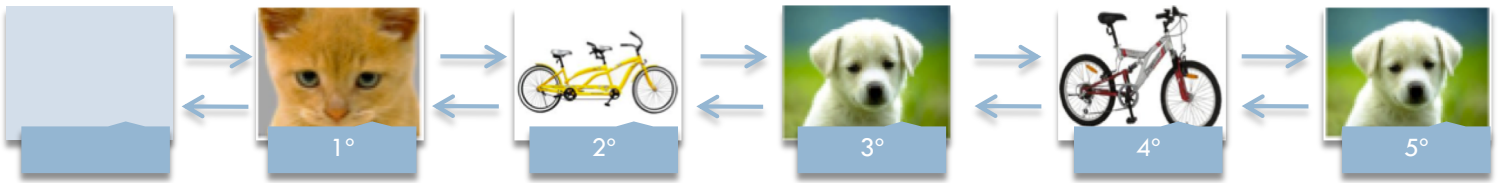
- As listas têm um método listIterator() que devolve um iterador especial, **`ListIterator<E>`**
  - Para além dos métodos do iterador **`Iterator<E>`**, acrescenta métodos que permitem iterar a lista em ambos os sentidos, para além de outras operações, como por exemplo a inserção e a substituição de elementos

Classe `LinkedList<E>`

- Em Java, **`LinkedList`** é uma estrutura que implementa (também) a interface **`List`**, sob a forma de lista duplamente ligada
- Eficiência das operações
  - Adicionar ou remover elementos no fim ou início da lista é eficiente
    - De referir que a lista ligada é uma estrutura mais dinâmica do que a lista em vector
  - Listar os elementos da lista de forma sequencial é eficiente
  - Acesso aleatório a elementos da lista não é eficiente




```
public class LinkedList<E> extends
  AbstractSequentialList<E> implements List<E> , ...
```



Classe `LinkedList<E>`

Métodos mais utilizados	
boolean add(E element)	Adiciona o elemento indicado no fim da lista
void add(int index, E element)	Adiciona na lista, na posição indicada o elemento
void addFirst(E element)	Adiciona o elemento indicado no início da lista
void addLast(E element)	Adiciona o elemento indicado no fim da lista
Object clone()	Devolve uma cópia *shallow* da lista
E get(int index)	Devolve o elemento que se encontra na lista na posição indicada
E getFirst()	Devolve o primeiro elemento da lista
E getLast()	Devolve o último elemento da lista
E remove(int index)	Remove e devolve o elemento que se encontra na lista na posição indicada
E removeFirst()	Remove e devolve o primeiro elemento da lista
E removeLast()	Remove e devolve o último elemento da lista
void clear()	Remove todos os elementos da lista
E set(int index, E element)	Substitui o elemento que se encontra na lista na posição indicada pelo novo elemento
int size()	Devolve o número de elementos na lista