



iscte

INSTITUTO
UNIVERSITÁRIO
DE LISBOA



emprego
digital

Módulo 1: Programação em Linguagem Java

Aula 12

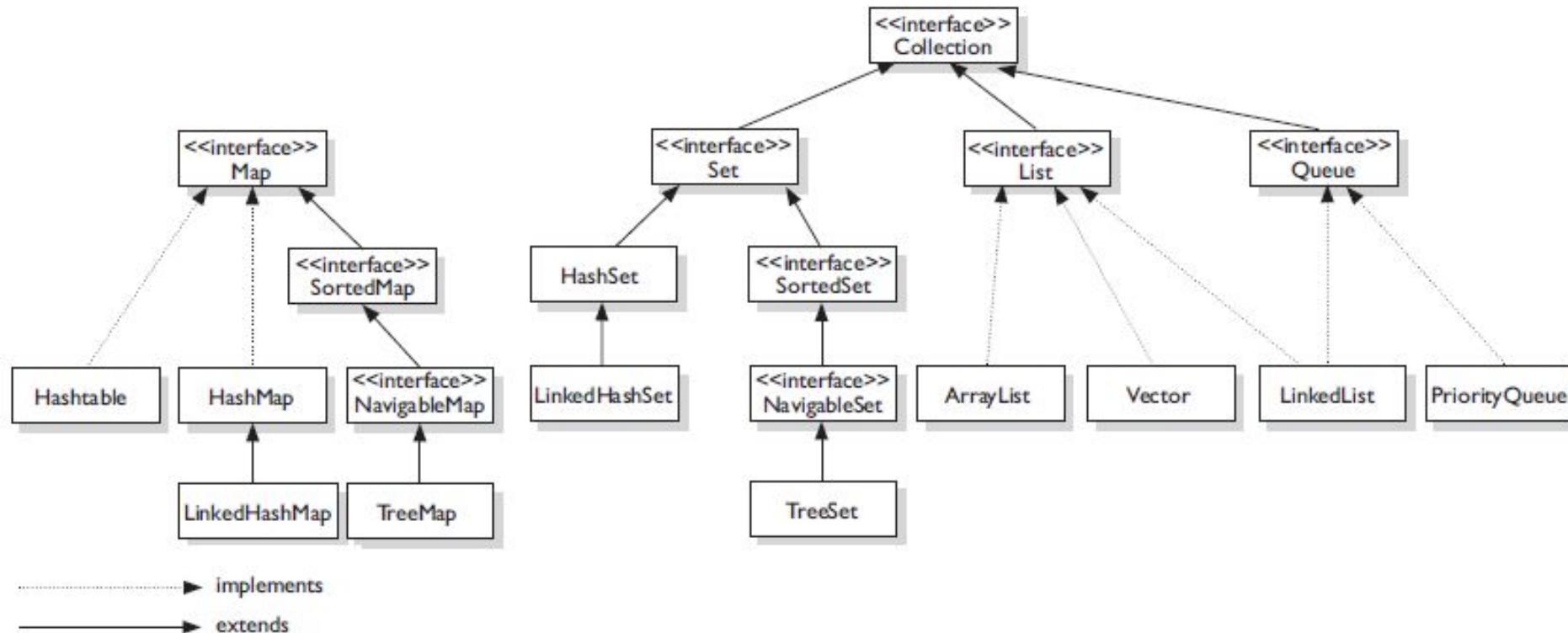
Java Collections Framework



O que é o Java Collections Framework

- O Java Collections Framework (JCF) é uma biblioteca de Java que disponibiliza recursos para trabalhar com **conjuntos de elementos** (coleções).
- Esta biblioteca inclui um grande conjunto de classes e interfaces: Listas, Conjuntos, Mapas, Arvores, etc.
- A ideia é os utilizadores poderem utilizar estas ferramentas para guardar dados, sem a preocupação de estar a implementar as estruturas de dados, já que a biblioteca fornece as estruturas de dados mais comuns.
- Cada estrutura de dados é útil para um propósito diferente, iremos conhecer as mais comuns nesta aula.

Classes do JCF



List<E> - Listas em Java

A interface List<E> define os métodos que caracterizam o comportamento de uma Lista.

Nota: A interface não implementa qualquer método, apenas define o comportamento!

Métodos:

`add(element)` – adiciona elemento à lista

`remove(element)` – remove elemento da lista

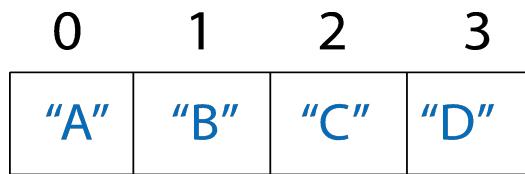
`contains(element)` – verifica se a lista contém o elemento passado

`get(index)` – obtém o elemento na posição index da lista

Que tipos de **Lista** existem em **Java**?

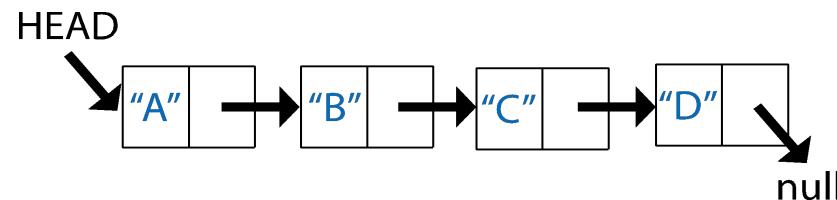
ArrayList

```
List<String> lista = new ArrayList<String>();  
lista.add("A");  
lista.add("B");  
lista.add("C");  
lista.add("D");
```



LinkedList

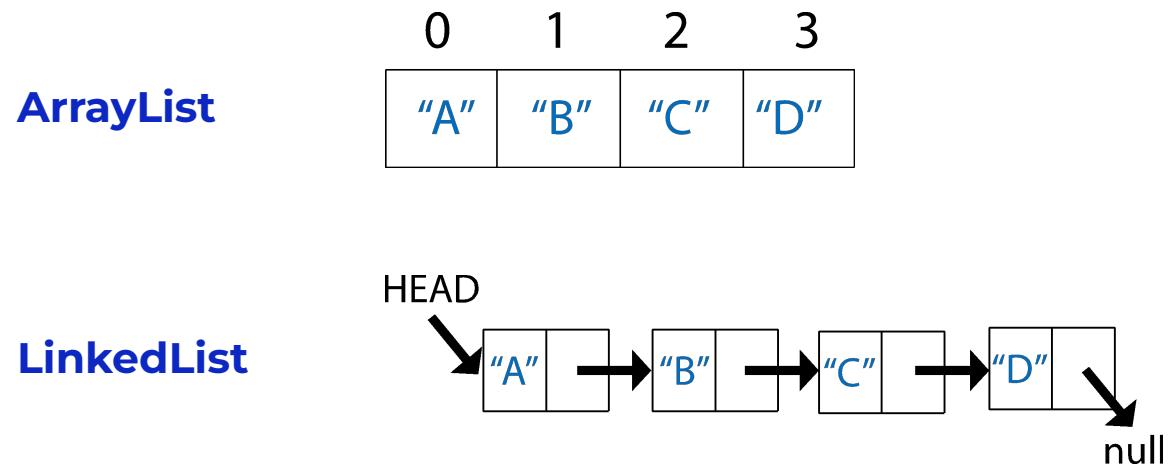
```
List<String> lista = new LinkedList<String>();  
lista.add("A");  
lista.add("B");  
lista.add("C");  
lista.add("D");
```



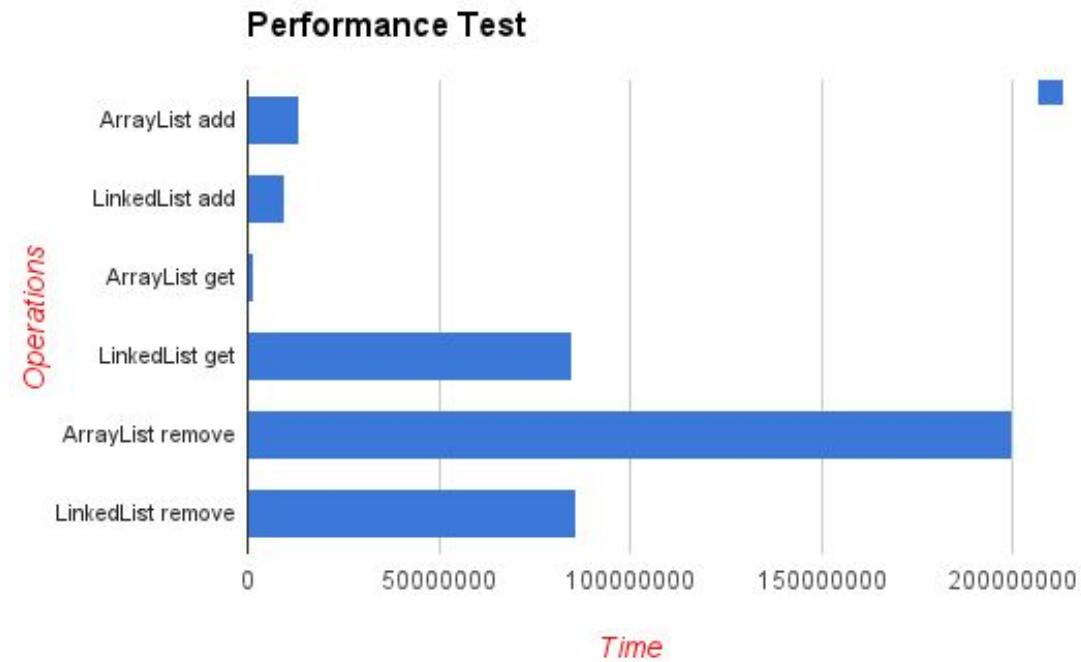
LinkedList vs ArrayList

Comparação simples:

- Remoção de um elemento no início da Lista



LinkedList vs ArrayList



Como percorrer uma **Lista**

- Independentemente da sua implementação (`ArrayList<>` ou `LinkedList<>`), as listas são particularmente úteis para guardar dados que depois podem ser percorridos.
- As listas assemelham-se a `Arrays`. Tal como nos `Arrays`, podemos utilizar um ciclo (`while`, `for`, etc):

```
ArrayList<Estudante> estudantes = new ArrayList<Estudante>();
for(int i=0; i<estudantes.size(); i++){
    System.out.println(estudantes.get(i).numero);
}
```

Como percorrer uma **Lista**

- Existe, no entanto, um outro tipo de ciclo for, o foreach.
- O foreach simplifica a iteração de uma lista.

```
ArrayList<Estudante> estudantes = new ArrayList<Estudante>();
```

```
    Tipo de          valor          lista a  
    dados           assumido       percorrer  
for(Estudante estudante : estudantes){  
    System.out.println(estudante.getNumero());  
}
```

Exercício 1

- Crie um tipo de objeto Contacto que guarda as seguintes informações:
 - Nome
 - Número de telefone
 - Email
- Crie uma classe de teste e na função main() crie uma lista (ArrayList<>) de Contactos. Experimente a inserção, remoção e obtenção de um contacto.
- Percorra toda a lista de contactos e imprima para o ecrã as seguintes informações:

João Silva - 910000000 - joao.silva@gmail.com

Pedro Fernandes - 910000001 - pedro.fernandes@gmail.com

Maria Rita - 910000002 - maria.ritinha@gmail.com

Queue<E> - Filas em Java

A interface Queue<E> define os métodos que caracterizam o comportamento de uma fila.

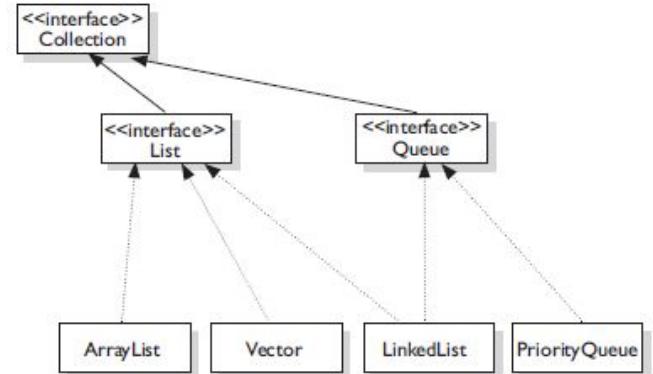
Nota: A interface não implementa qualquer método, apenas define o contrato.

Métodos:

`add(element)` – adiciona elemento à fila

`peek()` – mostra a cabeça da fila

`poll()` – remove e devolve o elemento no topo da fila



Implementação

- `LinkedList<E>` – A lista ligada também implementa o comportamento de uma fila
- `PriorityQueue<E>` – Guarda os elementos por ordem de prioridade. Necessário a classe E ser comparável ou passar um comparador ao construtor da `PriorityQueue<E>`

Exercício 2

- Pretende-se representar um gestor de senhas numa repartição de Finanças.
- Crie um objeto do tipo Senha que guarda as seguintes informações:
 - N° de Senha
 - Nome de utilizador
 - NIF
- Crie uma classe de teste e na função main() crie uma fila prioritária (`PriorityQueue<>`), adicione várias senhas. Implemente o comparador na classe senha, de modo às senhas serem ordenadas pelo número de senha. Confirme que a fila prioritária devolve as senhas pela ordem correcta
- Dica: utilize os métodos `peek()` e `poll()`

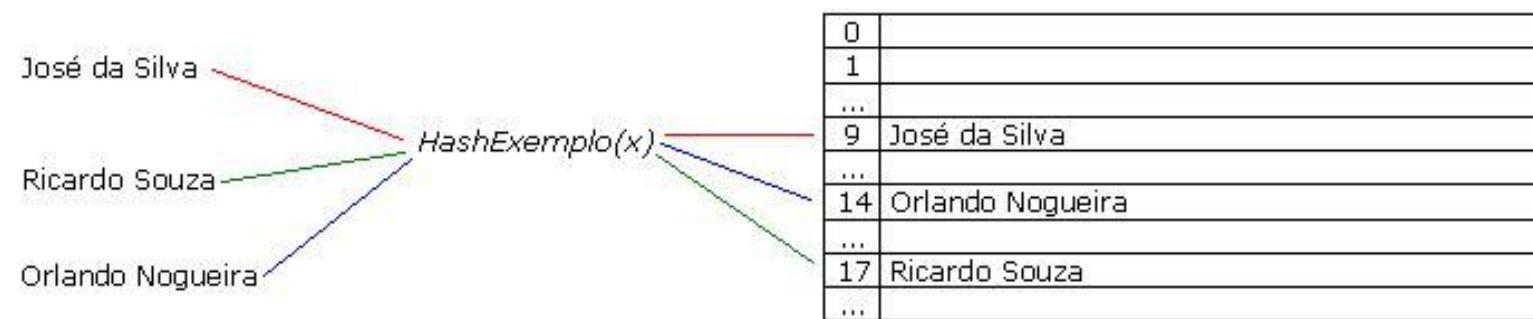
Conceito: Tabela de Dispersão

É calculado o hash() de cada objeto a inserir na coleção

- Hash nunca se deve repetir para objetos diferentes (seriam considerados iguais) e é uma função bastante difícil de implementar.
- Boa notícia: o IDE gera automaticamente a função hash(): Code | Generate | equals() and hashCode()

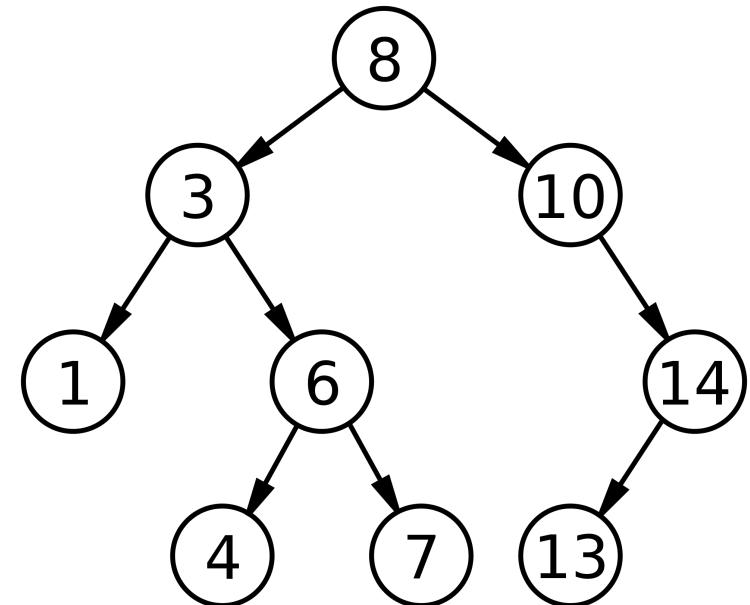
Pesquisa MUITO rápida... Cada objeto só tem um local possível. Assim, temos tempo constante para fazer uma pesquisa, sem depender do tamanho da coleção.

Eficiência máxima!!



Conceito: Árvore Binária de Pesquisa

- As árvores binárias são uma boa escolha para otimizar pesquisas.
- Novos elementos são guardados por comparação com os elementos já existentes.
- Necessário que elementos guardados sejam comparáveis ou que passemos um comparador ao construtor da Árvore
- Existem algumas colecções no JCF que guardam os dados em árvore: `TreeSet<E>` e `TreeMap<E>`, por exemplo



Set<> - Conjuntos em Java

Não guardam elementos repetidos

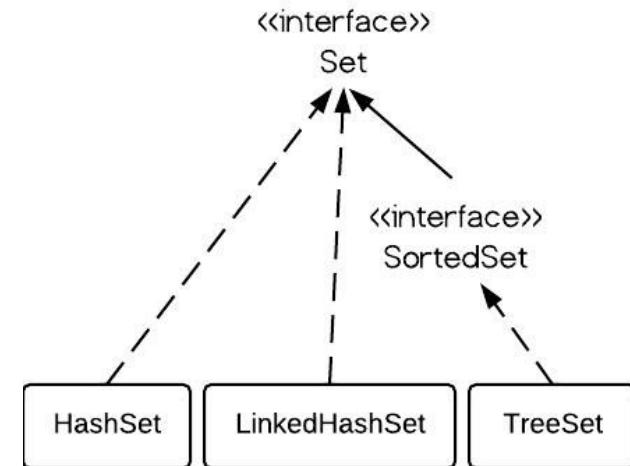
- Nem poderiam, porque utilizam o algoritmo de tabela de dispersão

Métodos:

`add(element)` – adiciona elemento ao conjunto

`contains(element)` – verifica se o conjunto contém o elemento dado

`remove(element)` – remove o elemento do conjunto



Nota: não existe o método `get()` pois não é esse o objectivo dos conjuntos.

Map<> - Mapas em Java

Os mapas permitem indexar chaves a valores. Um mapa funciona como uma tabela onde a cada chave corresponde um (e apenas um) valor.

Pesquisa muito eficiente, bastando ir à posição dada pelo hash() da chave na tabela de dispersão

Métodos:

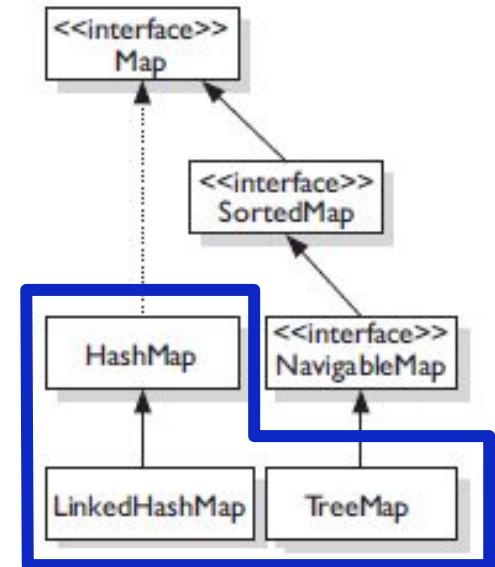
`put(chave, valor)` – Coloca o valor no mapa indexado pela chave

`get(chave)` – devolve o valor associado à chave

`containsKey(chave)` – verifica se o mapa contém algum valor para aquela chave

`keySet()` – devolve um Set<> contendo todas as chaves

`values()` – devolve todos os valores



Map<> - Exemplo

```
CHAVE   VALOR           CHAVE   VALOR  
  { }   { }  
Map<String,Carro> mapaCarros = new HashMap<String,Carro>();
```

```
Carro carro1 = new Carro("34-AA-64");
```

```
mapaCarros.put(carro1.getMatricula(), carro1);
```

- Esta instrução devolveria o carro1 (valor no mapa para a chave)

CHAVE **VALOR**



CHAVE	VALOR
34-AA-64	carrol
(...)	

Existem muitos outros tipos de coleções para além dos abordados aqui...

É sempre útil pesquisar pela coleção mais eficiente para cada caso.

Exercício 3

- Crie um tipo de objecto **Carro** que guarda as seguintes informações:
 - Matrícula
 - Marca
 - Ano de Venda
- Crie uma classe de teste e na função main() crie um mapa (**HashMap<>**) de Carros, onde a chave é a matrícula de carros. Experimente a inserção, remoção e obtenção de um carro.

Exercício 4

- Pretende-se recriar a função `String getMes(int mes)` que devolve o nome do mês com base no seu número, mas desta vez aplicando os conhecimentos de HashMap que obtiveram nesta aula. Como faria?



O futuro profissional começa aqui

