



Aula 11

Java Collections Framework

Iniciativa Conjunta:



Com o apoio de:

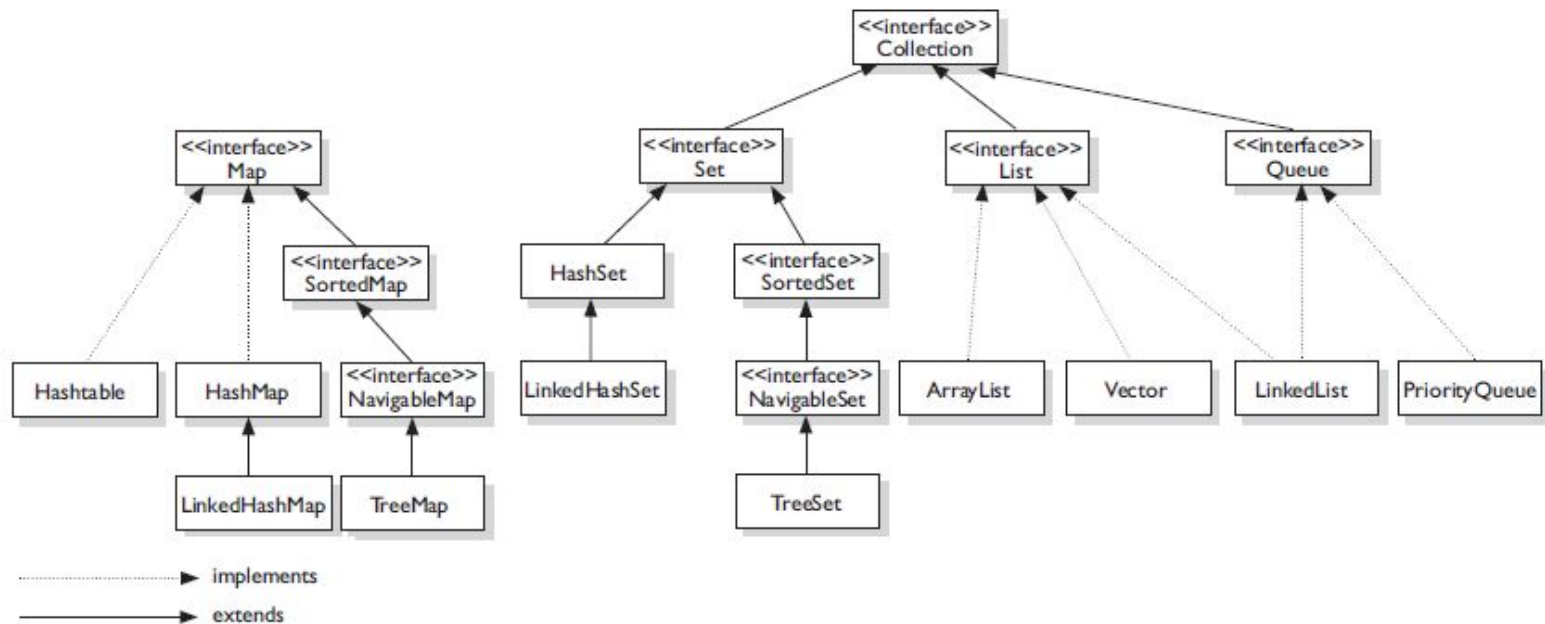




O que é o Java Collections Framework

- 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, Árvores, 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<> - Listas em Java

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

NOTA: A INTERFACE NÃO IMPLEMENTA QUALQUER MÉTODO!

Métodos:

`add(element)` – adiciona elemento à lista

`remove(element)` – remove elemento da lista

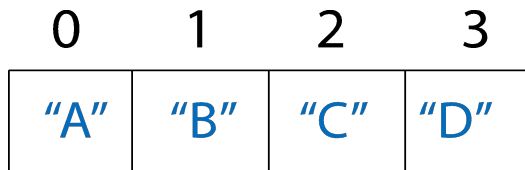
`contains(element)` – verifica se a lista contém o elemento passado

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

Que tipos de Lista existem em Java?

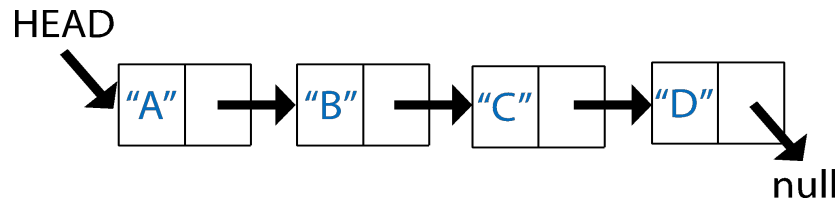
ARRAYLIST

```
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");
```

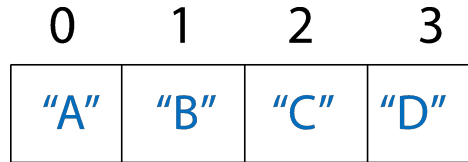


LinkedList VS. ArrayList

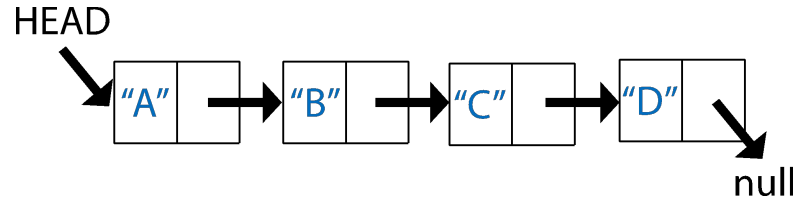
Comparação simples:

- Remoção de um elemento no início da Lista

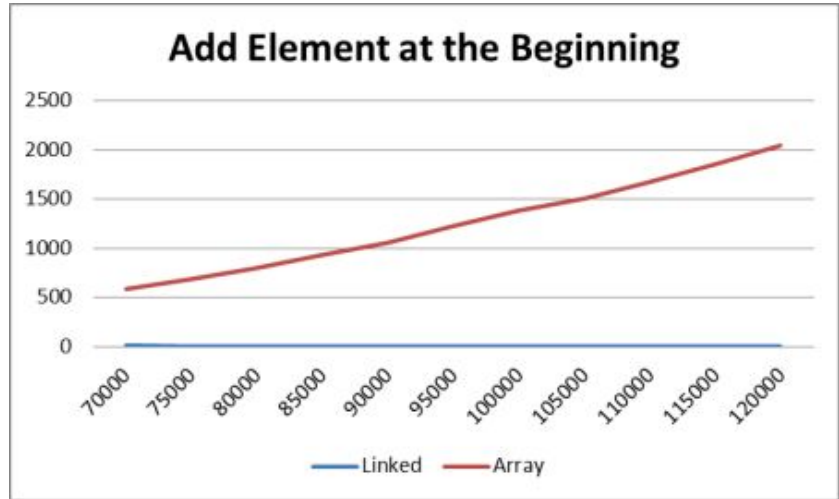
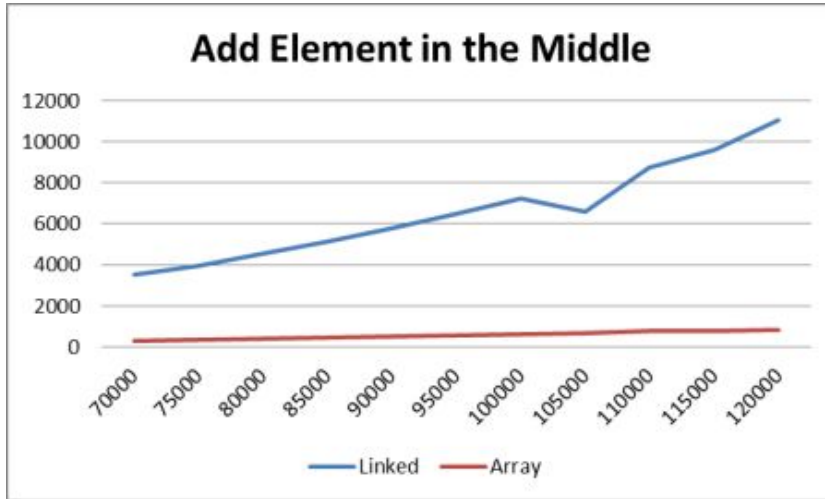
ARRAYLIST



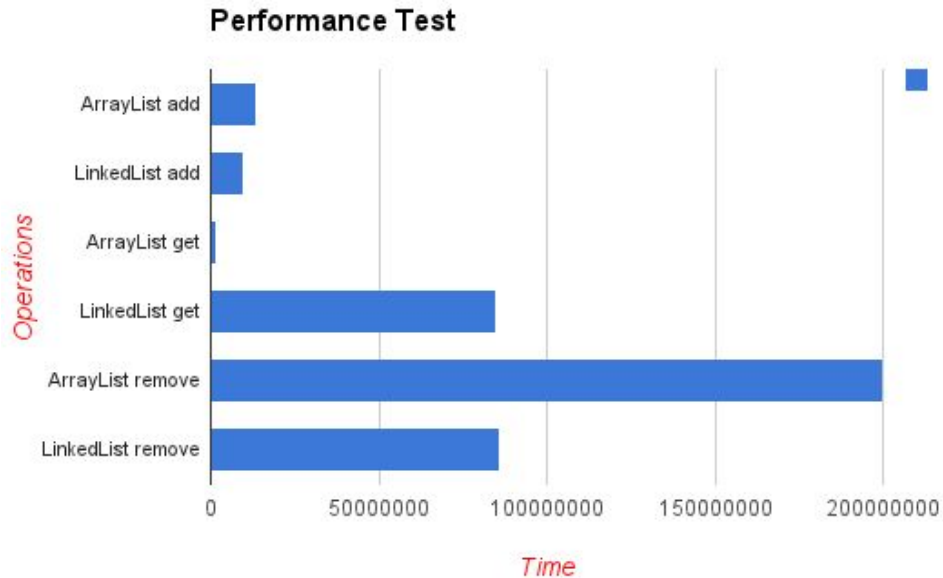
LINKEDLIST



LinkedList VS. ArrayList



LinkedList VS. ArrayList



Queue<> - Filas em Java

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

NOTA: A INTERFACE NÃO IMPLEMENTA QUALQUER MÉTODO!

Métodos:

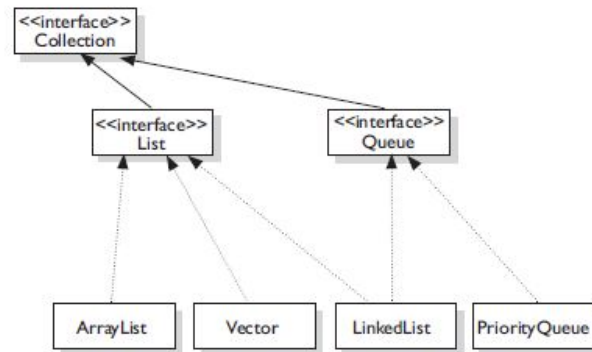
add(element) – adiciona elemento à fila

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 classe E ser comparável ou passar um comparador ao construtor da PriorityQueue<>

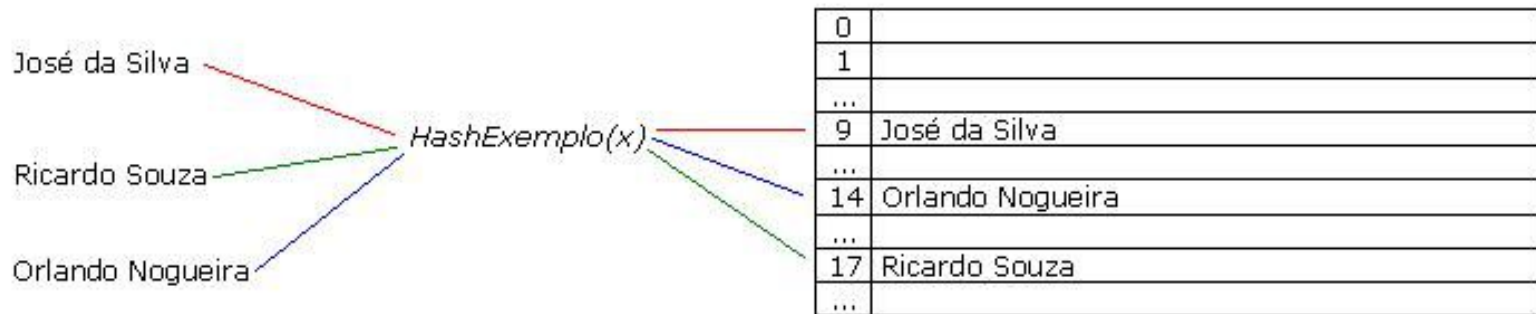


Conceito: Tabela de Dispersão

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!!!**



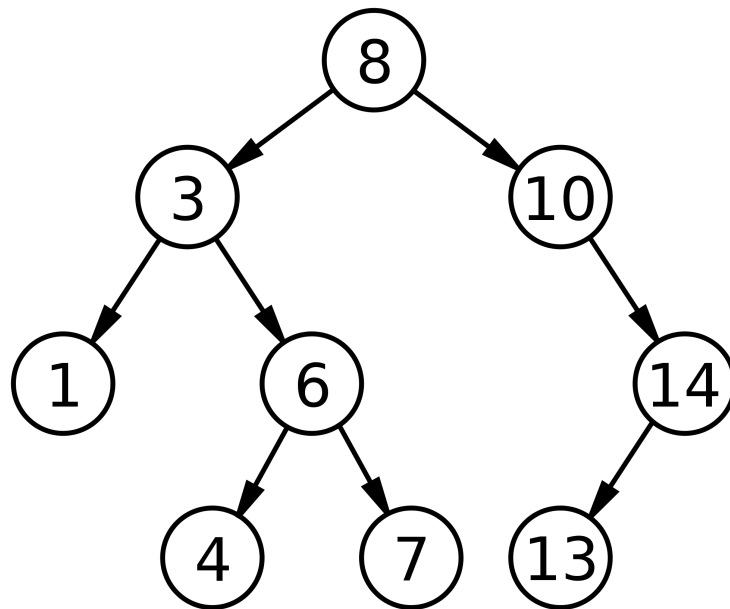
Conceito: Árvore Binária de Pesquisa

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

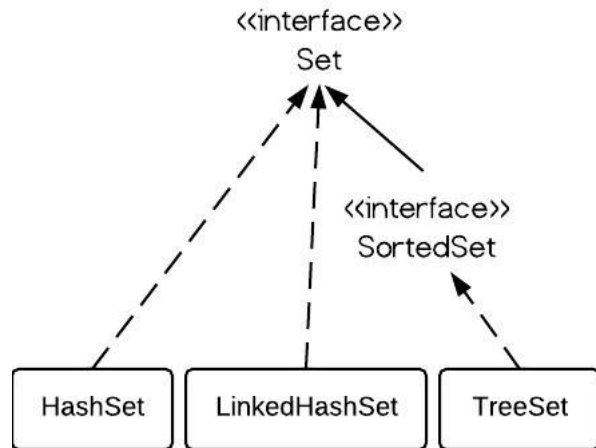
Métodos:

`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:

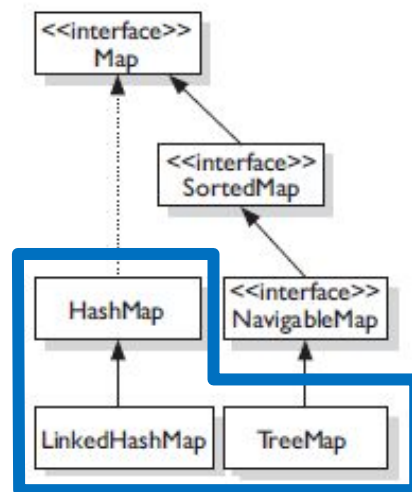
put(chave, valor) - Coloca o valor no mapa indexado pela chave

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


`Map<String,Carro> mapaCarros = new HashMap<String,Carro>();>`

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


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


`Carro c = mapaCarros.get("34-AA-64");`

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

CHAVE	VALOR
34-AA-64	carro1
(...)	



Conselhos

- Existem muitos outros tipos de coleções para além dos abordados aqui...
- **É sempre útil pesquisar pela coleção mais eficiente pois cada caso é um caso**



Exercício 1

- Crie um tipo de objecto 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 - 9100000000 - joao.silva@gmail.com  
Pedro Fernandes - 9100000001 - pedro.fernandes@gmail.com  
Maria Rita - 9100000002 - maria.ritinha@gmail.com
```




Exercício 2

- 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. Exprimente a inserção, remoção e obtenção de um carro.



Exercício 2

- Pretende-se representar um gestor de senhas numa repartição de Finanças.
- Crie um objecto 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())