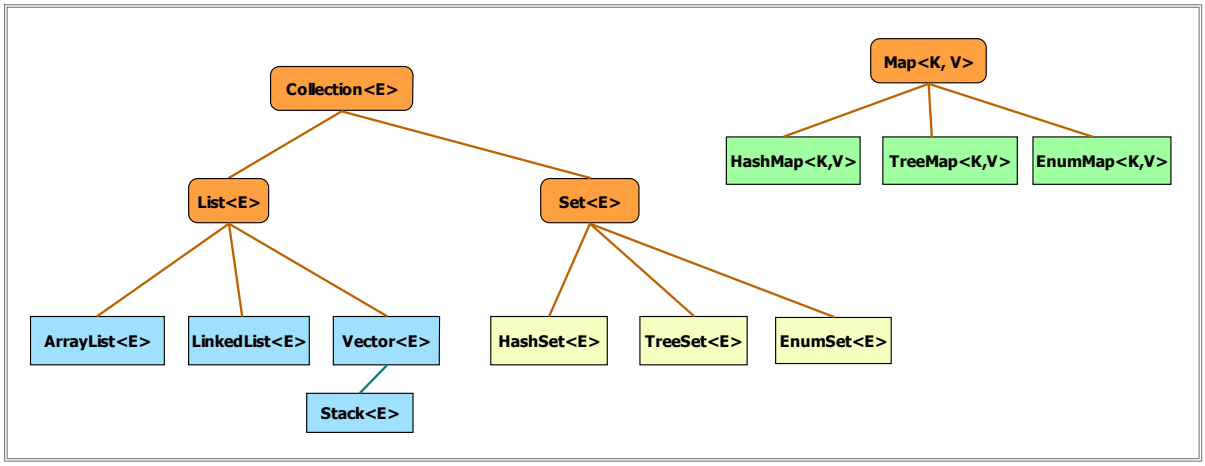
FICHA PRÁTICA 5

LABORATÓRIO DE COLECÇÕES

LIST<E>, SET<E>, MAP<K,V>

SÍNTESE TEÓRICA

O JFC ("JAVA5 Collections Framework") é uma sofisticada arquitectura de classes e tipos (interfaces) de JAVA, que, em síntese, oferece ao utilizador três **tipos** de organizações parametrizadas de objectos: *listas* (do tipo List<E>), *conjuntos* (do tipo Set<E>) e *correspondências* (do tipo Map<K, V>), e, para cada tipo, diversas classes concretas de implementação, tal como se apresenta na figura seguinte.



Tipos de Colecções e Classes de Implementação

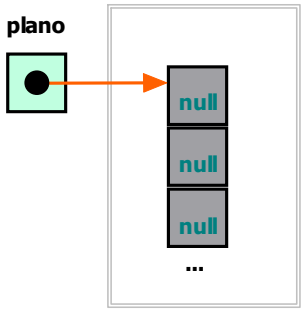
A colecção **ArrayList<E>** foi já apresentada e usada anteriormente, podendo passar a ser agora livremente utilizada nos exemplos seguintes com colecções.

As colecções do tipo **Set<E>** são implementações de *conjuntos de objectos*, pelo que os seus elementos não são indexados ou acessíveis por índice, nem podem ocorrer em duplicado (cf. noção matemática de conjunto). Das três implementações propostas no JCF vamos de momento centrar a nossa atenção em **HashSet<E>** e em **TreeSet<E>**. A colecção HashSet<E> implementa conjuntos usando uma *tabela de hashing* e *algoritmos de hashing*, enquanto que TreeSet<E> é uma implementação de conjuntos baseada numa *árvore binária ordenada*.

A declaração de um *hashset* ou de um *treeset* concreto, consiste em definir-se qual o tipo concreto para o seu tipo parâmetro, numa declaração normal mas paramétrica, onde, em geral, usando um **construtor**, é criada imediatamente uma *colecção* inicialmente vazia (atenção ao **construtor** que também é parametrizado), com dimensão inicial ou não.

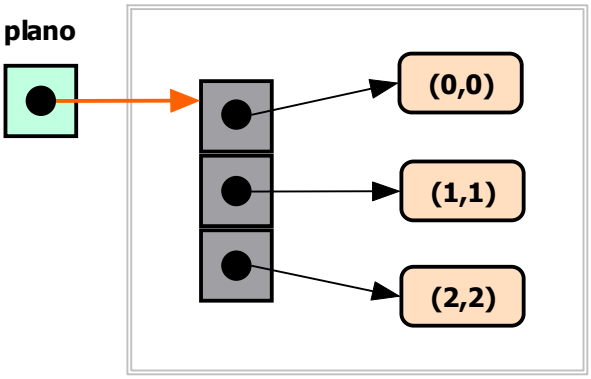
```
HashSet<String> codigos = new HashSet<String>();
HashSet<Ponto2D> caminho = new HashSet<Ponto2D>();
TreeSet<String> nomes = new TreeSet<String>();
TreeSet<Double> coordenadas = new TreeSet<Double>(500);
```

Qualquer colecção é, como vimos já, uma agregação de apontadores para objectos do tipo do parâmetro da colecção, que inicialmente tomam o valor **null**, já que nenhum elemento foi ainda inserido, cf.



HashSet<Ponto2D>

Posteriormente, e à medida que cada um dos Ponto2D for inserido, usando os diversos métodos que apresentaremos em seguida, cada posição referenciará uma instância de Ponto2D. Porém, nos conjuntos não há indexação, ou seja, não há acesso aos elementos por índice. Tal significa que a API de Set<E> é um subconjunto da API de List<E> sem as operações por índice.



HashSet<Ponto2D>

A API do tipo **Set<E>**, comum a ambas as implementações, é apresentada em seguida.
Todos os parâmetros representados como sendo do tipo *Collection* devem ser assumidos como podendo ser uma qualquer colecção de JAVA do tipo **List<E>** ou **Set<E>**.

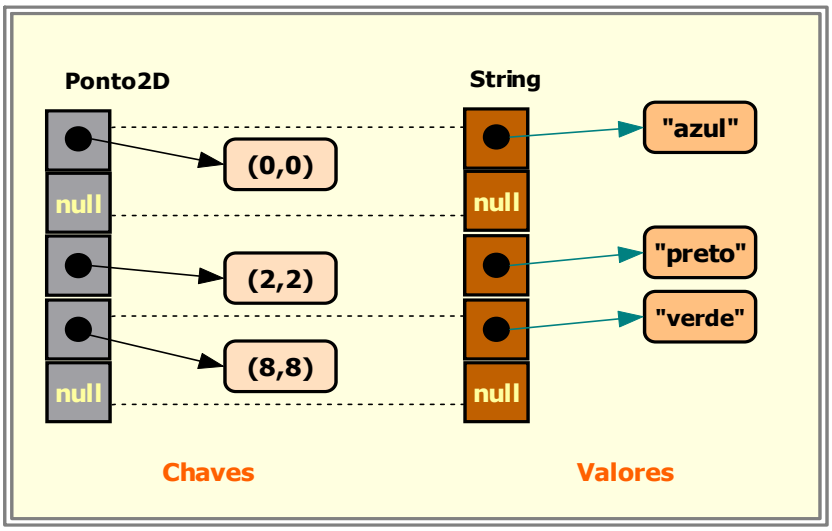
Categoria de Métodos	API de Set<E>
Inserção de elementos	`add(E o);` `addAll(Collection); addAll(int i, Collection);`
Remoção de elementos	`remove(Object o); remove(int index);` `removeAll(Collection); retainAll(Collection)`
Consulta e comparação de conteúdos	`boolean contains(Object o); boolean isEmpty();` `boolean containsAll(Collection); int size();`
Criação de Iteradores	`Iterator<E> iterator();`
Modificação	`clear();`
Subgrupo	`List<E> sublist(int de, int ate);`
Conversão	`Object[] toArray();`
Outros	`boolean equals(Object o); boolean isEmpty();`

Quadro – API de Set<E>

As estruturações de tipo **Map<K,V>** são correspondências finitas, um para um, entre os objectos de um tipo/classe **K** (chaves) e objectos do tipo/classe **V** (valores). Em informática é muito comum que a um dado código único se faça corresponder uma ficha de informação, por exemplo, a ficha de um aluno dado o seu código, a ficha de um produto dado o seu código, etc. Estas estruturações de informação correspondem a **Maps** de códigos para as respectivas fichas.

As duas principais classes de implementação do tipo **Map<K,V>** são **HashMap<K,V>** e **TreeMap<K,V>**, devendo as **TreeMaps** ser usadas sempre que pretendermos que as suas chaves se mantenham ordenadas (seja por um algoritmo de ordenação automático pré-definido de JAVA para esse tipo de chaves (exº String, Integer, etc.), seja através de um algoritmo de ordenação fornecido pelo programador ao respectivo construtor de TreeMaps (a ver mais tarde).

```
HashMap<Ponto2D, String> pixels =  new HashMap<Ponto2D, String>();
```



Exemplo de representação de um HashMap<Ponto2D, String>

A API do tipo **Map<K, V>**, comum a ambas as implementações, é apresentada em seguida.

Categoria de Métodos	API de Map<K,V>
Inserção de elementos	`put(K k, V v);` `putAll(Map<? extends K, ? extends V> m);`
Remoção de elementos	`remove(Object k);`

Consulta e comparação de conteúdos	`V get(Object k);`
	`boolean containsKey(Object k);`
	`boolean isEmpty();`
	`boolean containsValue(Object v); int size();`

Criação de Iteradores	`Set<K> keySet();`
	`Collection<V> values();`
	`Set<Map.Entry<K, V>> entrySet();`

Outros	`boolean equals(Object o); Object clone()`

Quadro – API de Map<K,V>

TreeMap<K, V> métodos adicionais
`TreeMap<K, V>()`
`TreeMap<K, V>(Comparator<? super K> c)`
`TreeMap<K, V>(Map<? extends K, ? extends V> m)`
`K firstKey()`
`SortedMap<K, V> headMap(K toKey)`
`K lastKey()`
`SortedMap<K, V> subMap(K fromKey, K toKey)`
`SortedMap<K, V> tailMap(K fromKey)`

Quadro – API parcial de TreeMap<K,V>

Algumas Regras Básicas de Utilização de Colecções Em Projectos:

- Escolher com critério se a colecção a criar deve ser uma lista ou um conjunto (duplicados ou não) ou então uma correspondência entre chaves e valores;
- Escolher para *sets* e *maps* uma classe de implementação adequada, cf**. Hash** (sem ordem especial) ou **Tree** (com comparação pré-definida ou definindo uma ordem de comparação);
- Nunca usar os métodos pré-definidos **addAll()** ou **putAll()**. Em vez destes, usar antes o iterador **for** e fazer **clone()** dos objectos a copiar para a colecção cópia (ver exemplos);
- Sempre que possível, os resultados dos métodos devem ser generalizados para os tipos List<E>, Set<E> ou Map<K,V> em vez de devolverem classes específicas como ArrayList<E>, HashSet<E> ou TreeSet<E> ou HashMap<K,V>. Aumenta-se assim a abstracção.

Exercícios:

Ex 1: Desenvolva uma classe Produto que represente a informação básica de um produto existente no armazém de uma dada empresa. Sobre cada produto pretende ter-se a seguinte informação:

```
private String codigo;   // código
private String nome;     // descrição
private int quant;       // quantidade em stock
private int alarme;      // mínimo de alerta – valor mínimo aceitável
```

Crie em seguida uma classe Empresa contendo o nome da empresa em questão e uma representação do stock da empresa, associando a cada código de produto a sua ficha de informação.
Para além dos construtores e métodos usuais, a classe Empresa deverá definir ainda os seguintes métodos de instância:

- Método que devolve todos os actuais códigos de produtos em stock;
- Método que insere um novo produto no stock da empresa;
- Método que remove do stock o produto de código dado;
- Método que altera a quantidade de um produto em stock de um valor dado;
- Método que devolve a quantidade total de produtos em stock;
- Método que verifica se o produto de código dado existe;
- Método que cria uma lista com os códigos dos produtos com quantidade <= alarme;
- Método que devolve uma cópia do stock;
- Método que devolve a informação de um produto de código dado;

Ex 2: Crie uma classe Paises que estabeleça uma correspondência entre o nome de um dado país e a informação sobre a sua capital (FichaDeCapital), designadamente: nome da cidade, população, número de veículos, salário médio (real) e custo de vida mensal médio (real).
Implemente os seguintes métodos na classe Paises:

- Determinar o número total de países;
- Devolver os nomes dos países com capitais com população acima de um valor dado;
- Dado o nome de um país, devolver ficha completa da sua capital;
- Alterar a população da capital de um dado país;
- Inserir a informação de um novo país;
- Criar uma listagem com os nomes de todas as capitais registadas;

- Determinar o somatório de todas as populações das capitais;
- Dada um Map de nome de país para FichaDeCapital, para cada país que exista na lista de países alterar a sua ficha de capital e para cada país novo inserir a sua informação.
- Dada um conjunto de nomes de países, remover as suas fichas de capital;

Ex 3: Considerando a classe ContaPrazo anteriormente desenvolvida, crie agora uma classe Banco que associe a cada código de conta uma ContaPrazo. A classe Banco deverá implementar métodos que realizem as seguintes operações:

- Inserir uma nova conta;
- Determinar o conjunto de códigos das contas pertencentes a dado titular;
- O mesmo que o anterior mas para um conjunto de nomes de titulares;
- Determinar os códigos das contas com capital superior a um valor dado;
- Criar um Map das contas com taxa de juro superior a um valor dado;
- Conjunto dos códigos das contas que vencem juros no dia de hoje;
- Dada uma lista de códigos de contas incrementar as suas taxas de juro de um valor X;
- Devolver os nomes de todos os titulares de contas;
- Criar um Map que associe a cada nome de titular existente o valor total do capital investido nas suas várias contas (use métodos auxiliares).

Ex 4: Cada e-mail recebido numa dada conta de mail é guardado contendo o endereço de quem o enviou, a data de envio, a data de recepção, o assunto e o texto do mail (não se consideram anexos, etc.). Crie uma classe MailMap que associe a cada endereço de envio todos os mails recebidos (cf. classe Mail) e implemente as seguintes operações:

- Determinar o total de endereços a partir dos quais se recebeu mail;
- Guardar um novo mail recebido;
- Determinar quantos mails têm por origem um dado endereço;
- Criar uma lista contendo todos os endereços que enviaram mails contendo no seu assunto uma lista de palavras dada como parâmetro;
- O mesmo que a questão anterior, mas criando um conjunto contendo os mails;
- Eliminar todos os e-mails recebidos antes de uma data que é dada como parâmetro;
- Criar uma lista dos endereços que hoje enviaram mails;
- Dada uma lista de palavras, eliminar todos os mails de um dado endereço que no seu assunto contenham uma qualquer destas (anti-spam);
- Eliminar todos os mails anteriores a uma data dada;
- Criar uma listagem com todos os endereços de mail oriundos de Portugal;