Map<K,V>

- Quando se pretende ter uma associação de um objecto chave a um objecto valor
- Na dimensão das chaves não existem elementos repetidos (é um conjunto!)
- Duas implementações disponíveis: HashMap<K,V> e TreeMap<K,V>
  - aplicam-se à dimensão das chaves as considerações anteriores sobre conjuntos

Percorrer uma colecção

- Podemos utilizar o foreach para percorrer a colecção toda, mas...
  - podemos querer parar antes do fim
  - podemos não ter o acesso à posição do elemento na colecção (caso dos conjuntos)
  - logo, é necessário um mecanismo comum para percorrer colecções

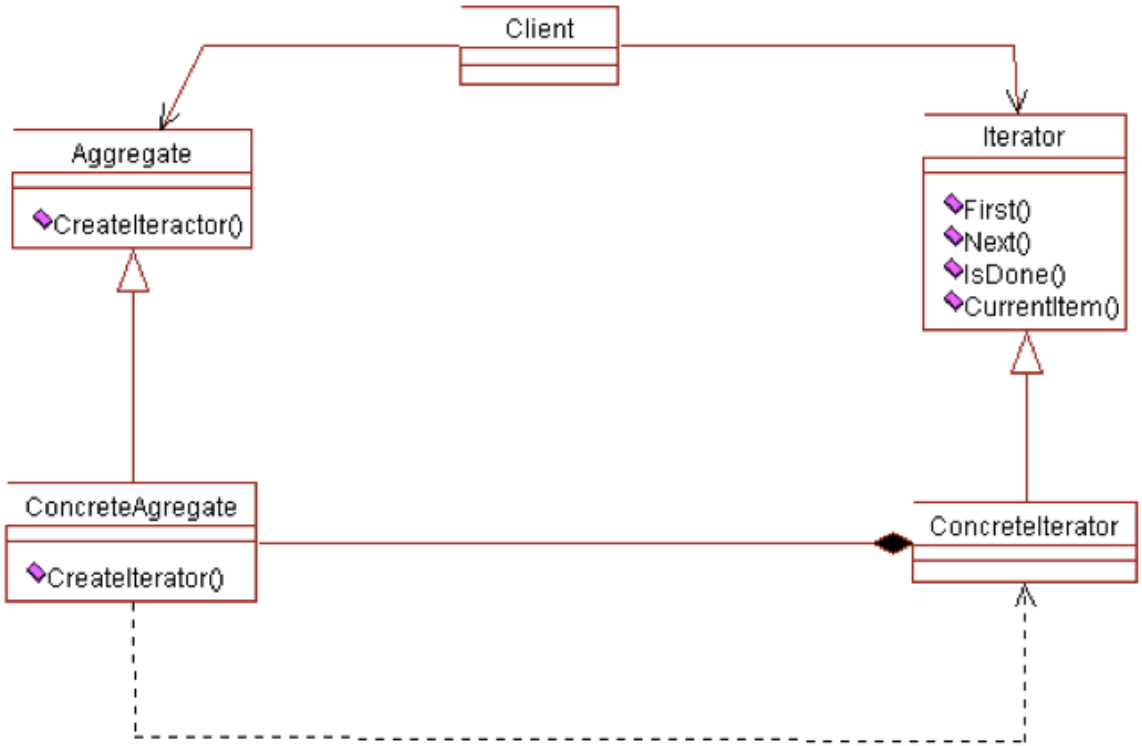
Iterator

- O Iterator é um padrão de concepção identificado e que permite providenciar uma forma de aceder aos elementos de uma colecção de objectos, sem que seja necessário saber qual a sua representação interna
  - basta para tal, que todas as colecções saibam criar um iterator

- Um iterador de uma lista poderia ser:

- o iterator precisa de ter mecanismos para:
- aceder ao objecto apontado
- avançar
- determinar se chegou ao fim

- A estrutura geral do iterator será:



OUTROS ITERADORES DE COLECÇÕES

```
Iterator<E> iterator();  // método que cria um Iterador sobre a colecção
                         // que recebeu a mensagem. Se a colecção tem
                         // objectos de tipo E o Iterador é de tipo E
```

- UM `Iterator<E>` É UMA ESTRUTURA COMPUTACIONAL QUE IMPLEMENTA UM ITERADOR SOBRE TODOS OS ELEMENTOS DA COLECÇÃO ORIGEM (SEM ORDEM PREDEFINIDA), USANDO OS MÉTODOS `hasNext()`, `next()` E `remove()`.

EXEMPLO 1: USO DE `iterator()` COM `while() { ... }`

```
Iterator<E> it = colecção.iterator();
E elem; // tipo dos objectos que estão na colecção
while(it.hasNext()) {
   elem = it.next();
     // fazer qq. coisa com elem
}
```

EQUIVALENTE A:

```
for(E elem : colecção) { … }
```

NOTA: Mas *foreach* não permite parar procuras!!

Assim, em algoritmos típicos de procura tem que se usar a construção baseada em `iterator()`, cf.

```
Iterator<E> it = colecção.iterator();   // criar o Iterator<E>
E elem ;  // variável do tipo dos objectos da colecção
boolean encontrado = false;
while(it.hasNext() && !encontrado) {
    elem = it.next();
    if(elem possui certa propriedade) encontrado = true;
}
```

Exemplo:

```
// encontrar o 1º ponto com coordenada X igual a Y
Ponto2D pontoCopia = null;
Iterator<Ponto2D> it = linha.iterator();   // criar o Iterator<E>
Ponto2D ponto ;  // variável do tipo dos objectos da colecção
boolean encontrado = false;
while(it.hasNext() && !encontrado) {
    ponto = it.next();
    if(ponto.getX() == ponto.getY()) encontrado = true;
}
if(encontrado) pontoCopia = ponto.clone();
```

Map<K,V>

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()`

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)`

Regras para utilização de colecções

- 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)

Regras para utilização de colecções

- 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
- 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>.
  - aumentando-se assim a abstracção