PROGRAMAÇÃO ORIENTADA PELOS OBJECTOS

Mapas e Conjuntos

Java Collections Framework

Agrupamento de colecções

- Arquitectura unificada para representação e manipulação de colecções
  - Todas as linguagens orientadas pelos objectos possuem tais arquitecturas
- Disponibilizam um conjunto de
  - Interfaces
  - Implementações
  - Algoritmos

Entidades num agrupamento de colecções

- Interfaces
  - Tipos abstractos de dados que representam as colecções
  - Permitem uma manipulação das colecções de forma independente da sua representação
- Implementações
  - Implementações concretas das interfaces das colecções
  - São estruturas de dados reutilizáveis
- Algoritmos
  - Métodos que realizam operações úteis sobre os elementos das colecções
  - São polimórficos, isto é, o mesmo método pode ser utilizado em implementações distintas da respectiva interface da colecção

Vantagens de utilização de um agrupamento de colecções

- Menor esforço de programação
- Melhoria da qualidade de programação
- Facilita a interoperabilidade
- Reduz o esforço na criação de uma nova API
- Potencia a reutilização de código

Java Framework

			Implementações			
			Resizable array	Linked list	Balanced tree	Hash table
Interfaces	Collection	Set				HashSet
		SortedSet			TreeSet	
		List ✓	ArrayList ✓	LinkedList ✓		
		Map				HashMap
		SortedMap			TreeMap	

Bancos, bancos, bancos,

'Sorry I can't give you any money.....but
I can arrange a nice trouble-free loan!'

Not anymore!

Contas bancárias

- Vamos desenvolver um programa que permita efectuar operações sobre contas bancárias
- Enquadramento geral
  - Temos apenas um banco
  - Existem contas à ordem e contas a prazo
  - Os clientes do banco podem ser titulares de várias contas

Caracterização das contas

- Conta à ordem
  - Uma conta à ordem caracteriza-se por um *código* (único), o *nome do titular*, a *data de abertura*, a *data do último movimento*, o *saldo*, bem como todos os *movimentos* realizados na conta
  - Os movimentos na conta são *depósitos* ou *levantamentos*, os quais têm uma *data* e *valor* associados
- Conta a prazo
  - Uma conta a prazo caracteriza-se por um *código* (único), o *nome do titular*, a *data de abertura*, a *data de início de contagem de juros*, a qual é atualizada sempre que os juros são calculados e adicionados ao capital, o *montante de capital depositado*, o *prazo para cálculo de juros* e a *taxa pré-definida*

Contas à ordem no banco

- Gestão de contas à ordem
  - Criar uma conta
  - Encerrar uma conta
  - Inserir uma transação
  - Obter as contas pertencentes a dado titular
  - Obter as contas pertencentes a um grupo de titulares
  - Obter as contas com saldo superior a um dado valor
  - Listar os nomes de todos os titulares de contas

Contas a prazo no banco

- Gestão de contas a prazo
  - Criar uma conta
  - Encerrar uma conta
  - Obter as contas pertencentes a um titular
  - Obter as contas pertencentes a um grupo de titulares
  - Obter as contas com capital superior a um dado valor
  - Obter as contas com taxa de juro superior a um dado valor
  - Obter as contas que vencem juros no dia de hoje
  - Listar os nomes de todos os titulares de contas
  - Obter os saldos das contas com juros vencidos de um titular

Conta à ordem

- Operações sobre a conta à ordem
  - Criar uma conta
  - Calcular o número de dias passados desde a abertura da conta
  - Registar uma transação, de depósito ou de levantamento
  - Encerrar a conta, indicando o respectivo saldo

Conta a prazo

- Operações sobre a conta a prazo
  - Criar uma conta
  - Calcular o número de dias passados desde a abertura da conta
  - Alterar a taxa de juros
  - Actualizar o vencimento de juros
    - Alcançado o prazo para juros, calcula os juros, junta ao capital e regista a nova data de cálculo de juros
  - Verificar se hoje é o dia de calcular os juros
  - Encerrar a conta, calculando o valor total a pagar ao seu titular

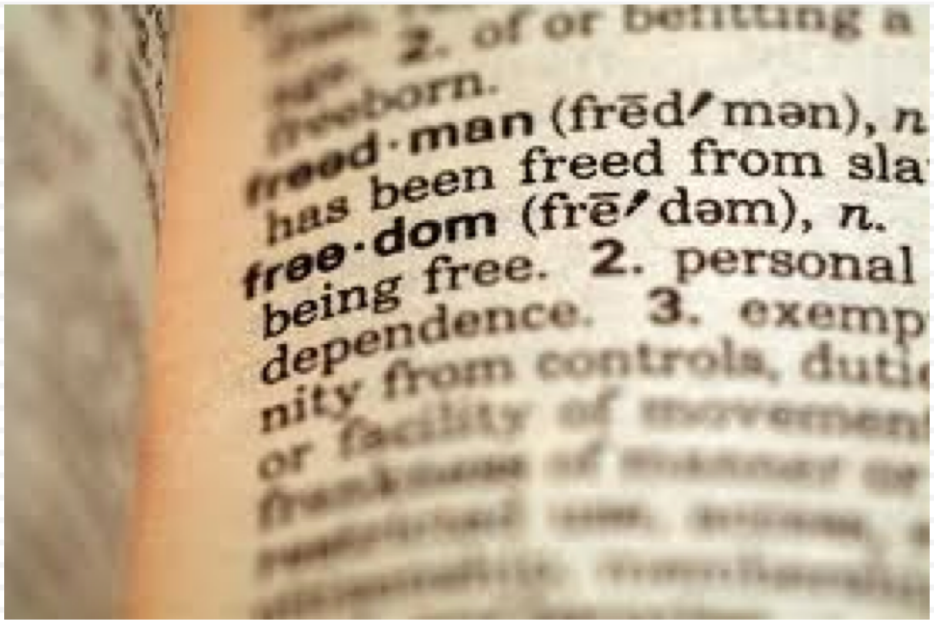
Proposta de solução para o problema

10 minutos

Um "pequeno" problema com as listas

- Se considerarmos que existem milhares de contas no banco, e guardarmos as contas numa lista, qual a ordem de grandeza do tempo necessário para obter informação sobre determinada conta?
  - Apesar de a interface list especificar um método contains, que permite verificar se determinado elemento está na lista ou não, esta solução é bastante ineficiente
  - A lista é especializada obter elementos por posição mas a abstracção que pretendemos usar agora é diferente...
- E se o problema fosse agora pesquisar palavras num **dicionário**?
  - A consulta não é por posição, mas sim por chave!
- A solução passa por utilizar *Mapas* implementados em estruturas apropriadas para aceder a colecções por conteúdo
  - Ex: tabelas de dispersão (*hash tables*), que serão estudadas detalhadamente na disciplina de Algoritmos e Estruturas de Dados

Dicionários



Interface `Map<K,V>`

- Um *Map*, estrutura definida em java.util, mapeia chaves a valores, com correspondências finitas e unívocas de um para um
  - Não existem chaves duplicadas
  - Cada chave pode mapear no máximo um único valor
  - Para que a pesquisa seja eficiente, o objecto correspondente a uma chave deve implementar os seguintes métodos:
    - hashCode()
    - equals()

```
public interface Map<K,V>
```

K, o tipo das chaves **V**, o tipo dos valores mapeados

Interface `Map<K,V>`

- Operações fundamentais
  - Inserção de elementos
    - Colocar um par chave-valor
    - Colocar vários pares chave-valor
  - Remoção de elementos
    - Remover o valor associado a uma chave
  - Consulta e comparação de conteúdos
    - Devolver o valor associado a uma chave
    - Verificar se existe uma determinada chave
    - Verificar se está vazio ou não
    - Verificar se existe um determinado valor
    - Devolver a sua dimensão
  - Criação de iteradores, numa perspectiva de map enquanto colecção de pares
    - Conjunto de chaves
    - Colecção de valores
    - Conjunto de pares chave-valor

Interface `Map<K,V>`

Métodos	
V put (K key, V value)	Associa o valor à chave
V remove(Object key)	Remove o mapeamento para a chave, se existir
void clear()	Remove todos os mapeamentos do map
V get(Object key)	Devolve o valor associado à chave, ou null se não existir
boolean containsKey(Object key)	Devolve true se o mapa contém um mapeamento com a chave indicada
boolean isEmpty()	Devolve true se o mapa não contém mapeamentos

Interface Map<K,V>

… métodos	
boolean containsValue (Object value)	Devolve true se o map mapeia uma ou mais chaves ao valor indicado
int size()	Devolve o número de mapeamentos existentes no map
Set<K> keySet()	Devolve um conjunto com as chaves existentes no map
Collection<V> values()	Devolve uma colecção com os valores existentes no map
Set<Map.Entry<K,V>> entrySet()	Devolve um conjunto com os mapeamentos existentes no map

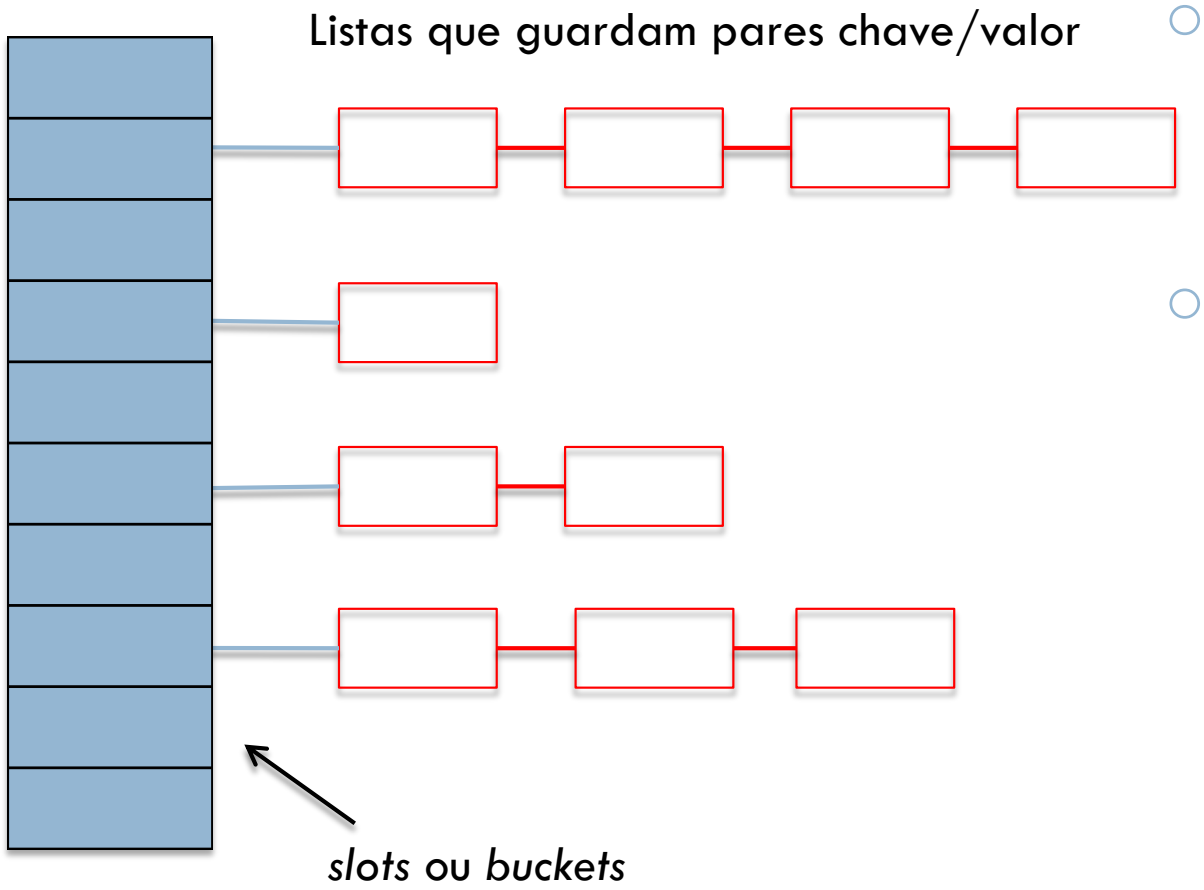
Interface `Map<K,V>`

- Algumas observações
  - É preciso ter algum cuidado quando se utilizam objectos mutáveis como chaves
    - Pode afectar comparações ao método equals
  - Por vezes este conceito é chamado de dicionário
    - No entanto, no contexto da linguagem Java, convém fazer a distinção entre a interface Map e a classe abstracta Dictionary, que Map actualmente substitui

Conceito de tabela de dispersão

- Estrutura sob a forma de vector de listas ligadas que permite aceder a informação através do respectivo conteúdo
- Para cada elemento, é possível calcular um código inteiro que permite inferir em que lista é que o elemento se encontra
  - Denomina-se função de *hashing*, permitindo associar uma chave ao seu valor (ex: nome de uma pessoa ao seu número de telefone)
- A eficiência depende de vários factores, como por exemplo o número de listas existentes versus o número de entradas na tabela
  - Estes factores são a capacidade inicial e o factor de carga

Tabela de dispersão



- Quantos slots?
  - Talvez 1.5 vezes o número de elementos que se espera guardar na tabelas, mas aproximando a um número primo
  - Quando a tabela atinge um factor de carga considerável, por exemplo 75% de slots preenchidos, então devemos reestruturar a tabela, aumentando a sua capacidade

Classe `HashMap<K,V>`

- A classe HashMap é uma implementação da interface Map baseado numa tabela de dispersão
  - Permite também null como valor e como chave
  - Suporta acesso eficiente
  - (Não garante ordem)

```
public class HashMap<K,V> extends AbstractMap<K,V>
                          implements Map<K,V>, ...
```

K, o tipo das chaves **V**, o tipo dos valores mapeados

Interface `SortedMap<K,V>`

- SortedMap é um Map que mantém as suas entradas em ordem ascendente, e de acordo com a ordem natural das chaves, ou então de acordo com o comparador que tenha sido fornecido aquando da sua criação
- Bastante utilizado em colecções ordenadas de pares chave-valor, como é o caso de dicionários e listas telefónicas
- Para além das operações de Map, permite ainda
  - Aplicar operações a zonas delimitadas do Map
  - Devolver a primeira e a última chave do mapa ordenado
  - Devolver o iterador que percorre os elementos em ordem ascendente
  - Devolver o comparador utilizado na ordenação, se este existir

```
public interface SortedMap<K,V> extends Map<K,V>
```

Interface SortedMap<K,V>

Métodos	
K firstKey()	Devolve a primeira chave (mais pequena) do mapa
K lastKey()	Devolve a última chave (maior) do mapa
SortedMap<K,V> headMap(K toKey)	Devolve uma vista para o mapa só com pares cuja chave é menor que toKey
SortedMap<K,V> subMap (K fromKey, K toKey)	Devolve uma vista para o mapa só com pares cuja chave é maior ou igual a fromKey e menor que toKey
SortedMap<K,V> tailMap(K fromKey)	Devolve uma vista para o mapa só com pares cuja chave é maior ou igual a fromKey
Comparator<? super K> comparator()	Devolve o comparador usado na ordenação (se existir)

Classe `TreeMap<K,V>`

- A classe TreeMap é uma implementação da interface SortedMap baseada numa árvore binária de pesquisa
  - Permite também null como valor e como chave
  - Também suporta acesso eficiente
  - Garante ordem

```
public class TreeMap<K,V> extends AbstractMap<K,V>
                          implements NavigableMap<K,V>, ...
```

K, o tipo das chaves **V**, o tipo dos valores mapeados

Conjuntos

Interface `Set<E>`

- Um set modela o conceito matemático de conjunto
  - Ex: baralho de cartas
- A interface Set estende Collection com as seguintes restrições
  - Não são admitidos elementos duplicados
  - Não estabelece ordem entre elementos
- A interface não adiciona métodos para além dos que são herdados de Collection

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

Interface `Set<E>`

Métodos	
boolean add(E e)	Adiciona o elemento indicado ao conjunto se ainda não existir no conjunto
boolean remove(Object o)	Remove o elemento indicado do conjunto, se este existir no conjunto
void clear()	Remove todos os elementos do conjunto
boolean contains (Object o)	Devolve true se o conjunto contém o elemento indicado
boolean isEmpty()	Devolve true se o conjunto não contém elementos
int size()	Devolve o número de elementos do conjunto
Iterator<E> iterator()	Devolve um iterador sobre os elementos do conjunto

Classe `HashSet<E>`

- A classe HashSet é uma implementação geral e eficiente da interface Set, baseada numa tabela de dispersão
  - Permitem também null como valor e como chave
  - Não garante a ordem de iteração ao longo do tempo

```
public class HashSet<E> extends AbstractSet<E>
                        implements Set<E>, ...
```

Interface `SortedSet<E>`

- SortedSet é um conjunto que permite estabelecer o conceito de ordem total entre os seus elementos
- Os elementos são ordenados de acordo com a sua ordem natural, através da implementação da interface Comparable, ou então de acordo com o comparador que tenha sido fornecido aquando da sua criação
- As novas operações que utilizam a ordenação são as seguintes:
  - Aplicar operações a zonas delimitadas do Set
  - Devolver o primeiro e o último elemento do conjunto ordenado
  - Devolver o iterador que percorre o conjunto por ordem ascendente
  - Devolver o comparador utilizado na ordenação, se este existir

```
public interface SortedSet<E> extends Set<E>
```

Interface SortedSet<E>

Métodos	
E first()	Devolve o primeiro elemento (mais pequeno) do conjunto
E last ()	Devolve o último elemento (maior) do conjunto
SortedSet<E> headSet(E toElem)	Devolve uma vista para o conjunto só com elementos menores que toElem
SortedSet<E> subSet(E fromElem, E toElem)	Devolve uma vista para o conjunto só com elementos maiores ou iguais a fromElem e menores que toElem
SortedSet<E> tailSet(E fromElem)	Devolve uma vista para o conjunto só com elementos maiores ou iguais a fromElem
Comparator<? super E> comparator()	Devolve o comparador usado na ordenação (se existir)

Classe `TreeSet<E>`

- A classe TreeSet é uma implementação geral e eficiente da interface SortedSet, baseada numa árvore binária de pesquisa
  - Permitem também null como valor e como chave
  - Garante a ordem de iteração ao longo do tempo

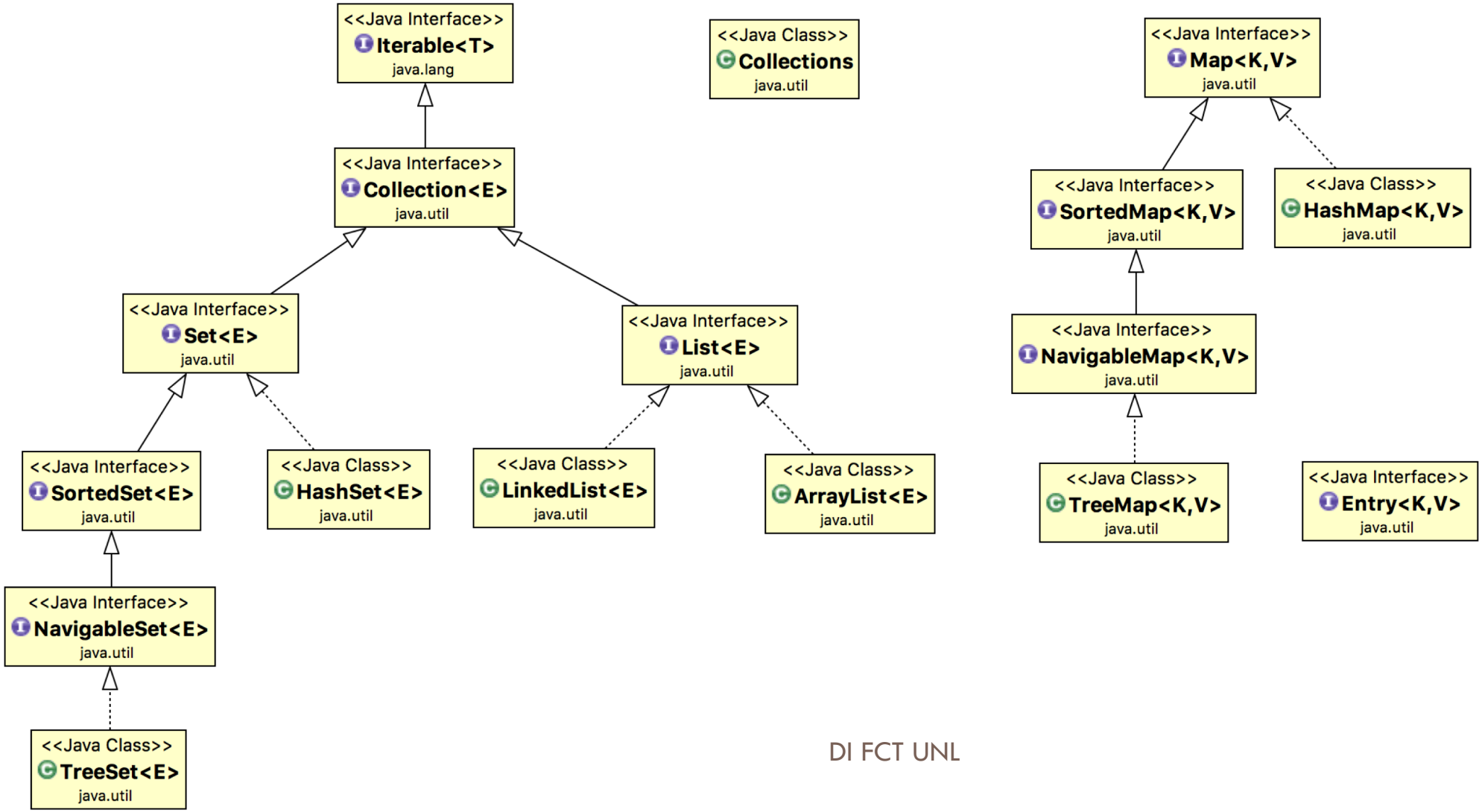
```
public class TreeSet<E> extends AbstractSet<E>
                        implements SortedSet<E>, ...
```

Java Framework

Java Framework

			Implementações			
			Resizable array	Linked list	Balanced tree	Hash table
Interfaces	Collection	Set				HashSet
		SortedSet			TreeSet	
		List ✓	ArrayList ✓	LinkedList ✓		
		Map				HashMap
		SortedMap			TreeMap	

Java Framework



Voltando ao exemplo do Banco

- Código ilustrado no eclipse