
Programação Orientada Pelos Objectos
Mapas e Conjuntos

2

# 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 (“escondem” a implementação)
  - Por norma formam uma hierarquia
- Implementações
  - Implementações concretas das interfaces das colecções
  - São essencialmente estruturas de dados reutilizáveis
- Algoritmos
  - Métodos que realizam operações úteis sobre os elementos das colecções
    - Ex: ordenar, pesquisar, copiar, etc.
  - 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 | |

# 7 Bancos, bancos, bancos, bancos


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
  - 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
  - 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

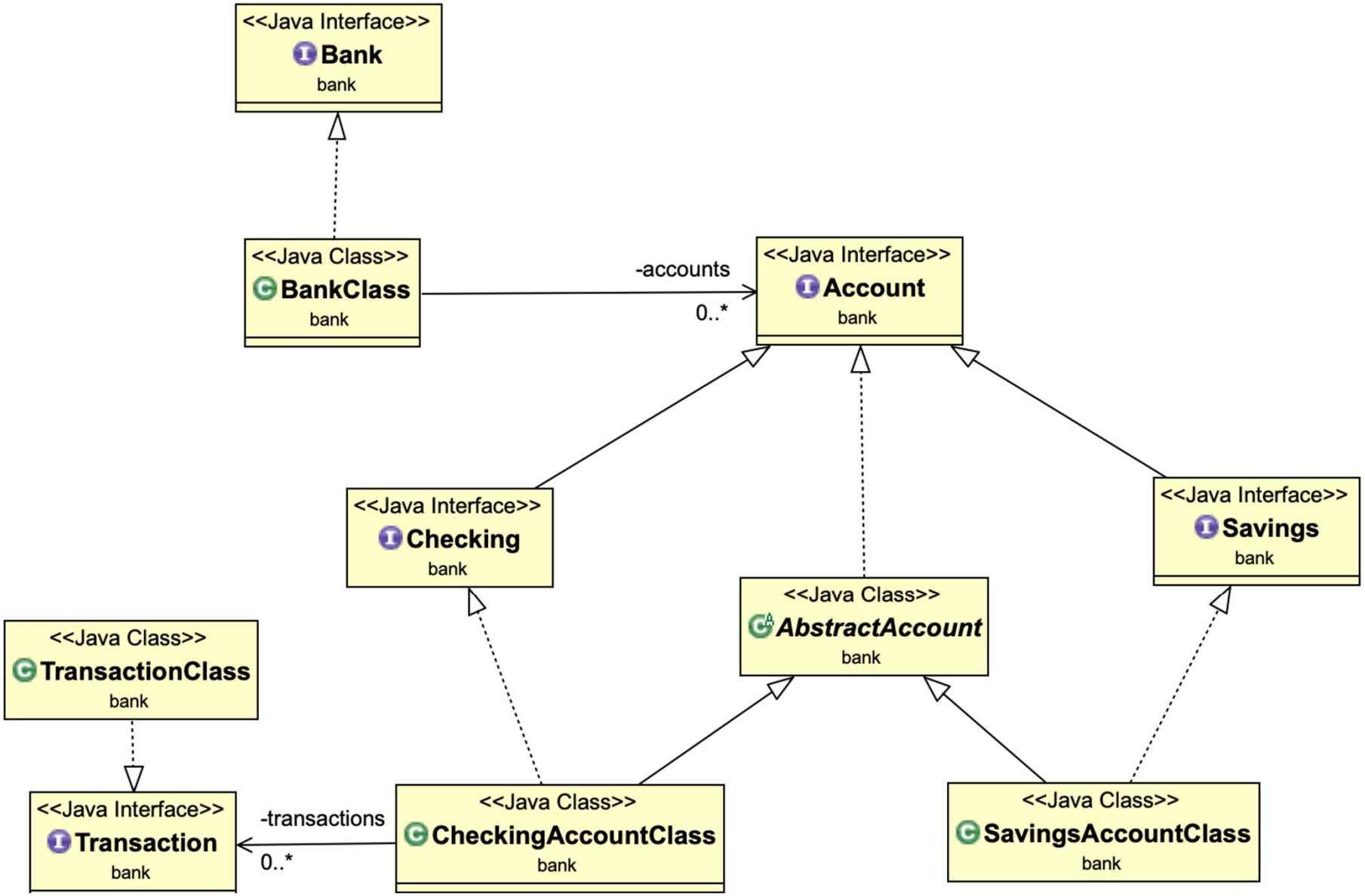
# Proposta de solução para o problema



10 minutos

# Proposta de solução para o problema



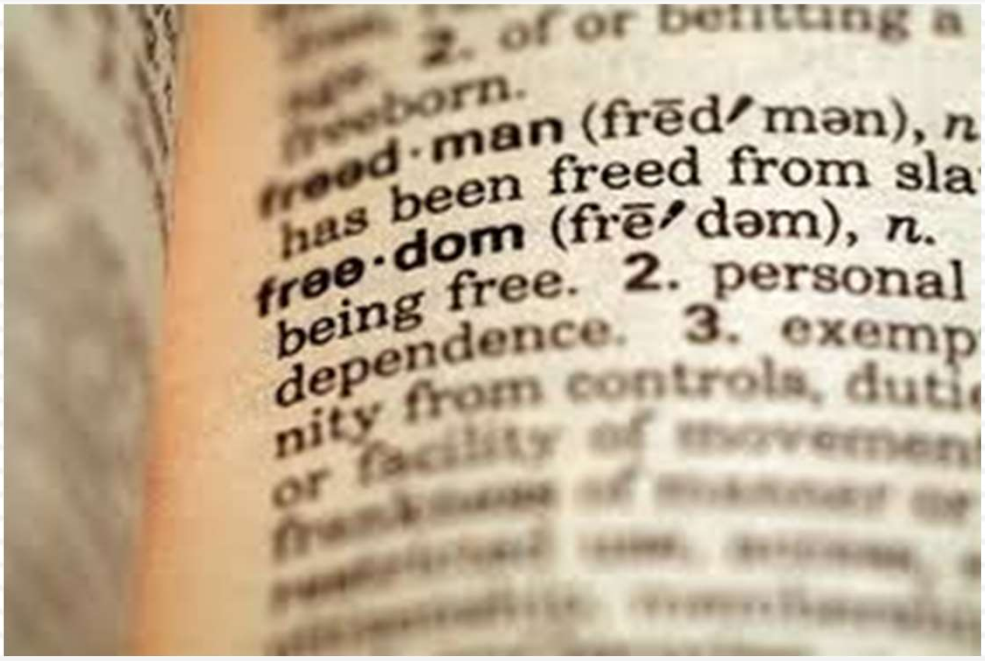


# Um “pequeno” problema com as listas



- 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 intrínsecas 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

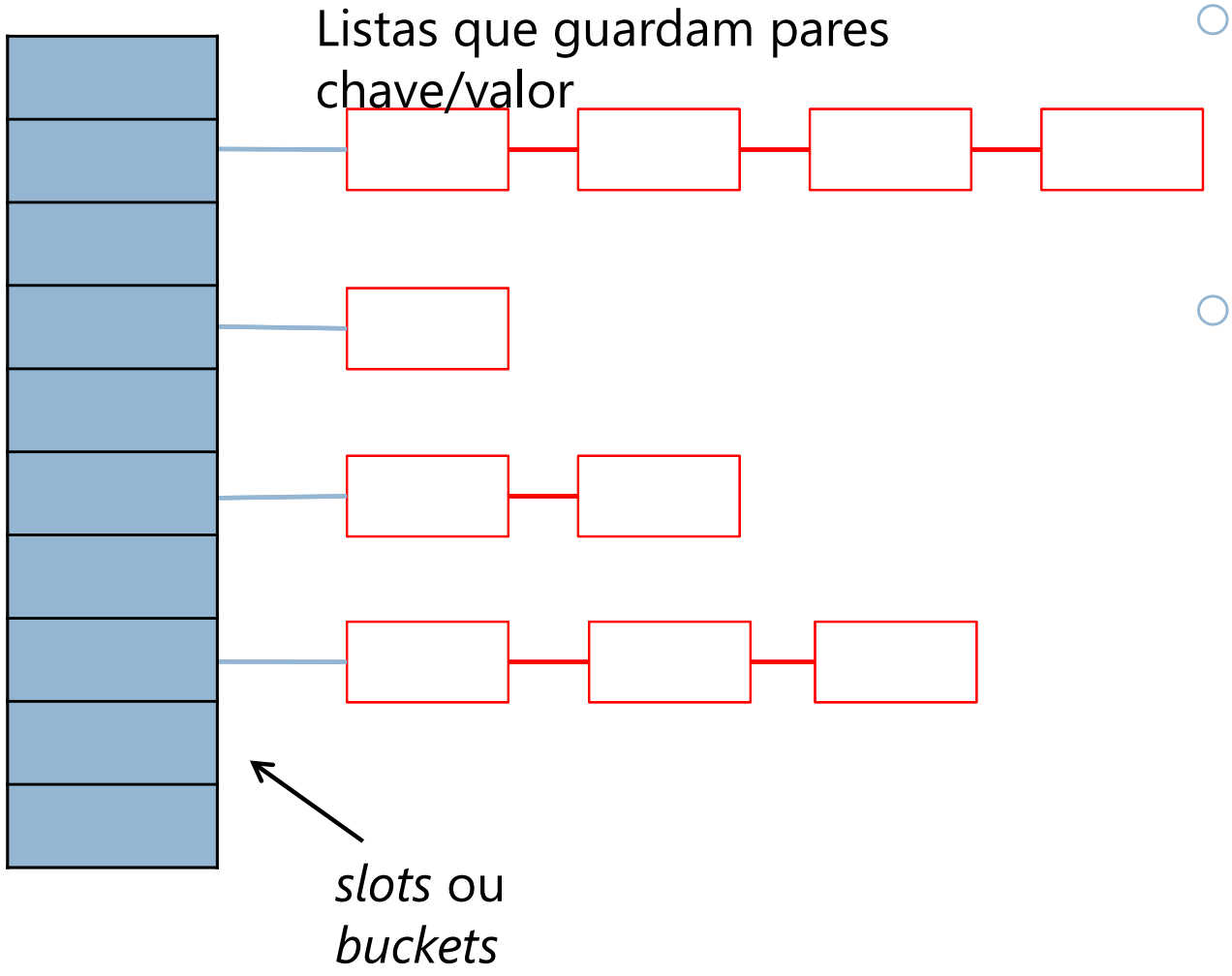
# Conceito de tabela de dispersão



- 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