Introdução à Programação

Usando Arrays para Implementar Coleções

Coleção

Procurar o ônibus escolar

Acrescentar um carro novo

Remover o táxi

Coleções

Tópicos da Aula

- Hoje, aprenderemos a utilizar **arrays** para implementar coleções
  - Conceito de coleção
  - Coleção como Tipo Abstrato de Dados
  - Usando arrays para implementar o tipo Coleção
  - Operações básicas de coleção
    - Inserção
    - Busca
    - Remoção
    - Existe
  - Duplicando capacidade de arrays
- Veremos, ainda, outros tópicos relacionados a arrays
  - Métodos retornando muitos objetos
  - Arrays bidimensionais

Coleções

- Uma **coleção** é um objeto que serve como repositório para outros objetos
- Uma coleção, geralmente, fornece serviços tais como: inserção, remoção, busca, além de outros serviços de gerenciamento dos objetos do repositório
- Coleções são muito utilizadas em sistemas
- A ordem dos elementos pode ou não ser importante em uma coleção
  - Depende da aplicação
- Elementos de uma coleção podem ser homogêneos ou heterogêneos
  - Elementos podem ser do mesmo tipo ou não

Abstraindo a Implementação de Coleção

- Coleções podem ser implementadas de várias formas diferentes
- A estrutura de dados que implementa uma coleção deve ser **abstraída** do programador que irá usá-la
  - Deve esconder os detalhes desnecessários
  - O que importa é o que uma coleção é capaz de fazer e não como
- Devemos separar a interface de coleção de sua implementação
  - Minimiza complexidade de trabalhar com coleções
  - Possibilita uma mudança de implementação da coleção

Coleção como um Tipo Abstrato de Dado

- Um **Tipo Abstrato de Dado** (TAD) é um conjunto organizado de informações e as operações que podem atuar sobre estas informações
  - Define um novo tipo de dado
- As operações definem a interface de um TAD
- Um código implementa corretamente um TAD desde que obedeça o que está sendo definido na interface do TAD
- O conceito de objeto é perfeito para criar TADs, pois os detalhes internos de implementação de um objeto são encapsulados
  - Classe é um TAD com uma implementação

Coleção como um Tipo Abstrato de Dado

- Podemos dizer que uma coleção é um TAD e sua interface deve ter pelo menos as seguintes operações:
  - Inserção
  - Remoção
  - Busca
  - Existe (Pertence)
- Podemos, então, escrever uma classe em Java que implemente o TAD **coleção**, utilizando arrays
  - Escreveremos um método para cada operação
- Na API de Java, existe várias implementações de coleção e a interface de coleção (Collection) possui vários métodos
- MiniJava possui a classe MiniJavaVector que é uma implementação de coleção

Implementando Cadastro de Contas (Coleção)

```
public class CadastroContas {
   private Conta [] contas;
   private int proxima;
   private int maximo;
   public CadastroContas() {
      maximo = 100;
      contas = new Conta[maximo];
      proxima = 0;
   }
```

Construtor não recebe o tamanho da coleção

A princípio, esta coleção pode armazenar 100 contas

Tamanho da coleção não precisa ser definido pelo utilizador, pois uma coleção deve poder crescer

Inserindo uma Conta

```
void cadastrar (Conta c){
   if (proxima == maximo){
    duplicarTamanhoColecao();
   }
   contas[proxima] = c;
   proxima = proxima + 1;
 }
```

Verifica se capacidade da coleção estourou

Método auxiliar que duplica capacidade da coleção

Buscando uma Conta

```
Conta procurar(String num) {
  int i = 0; boolean achou = false;
  while ((!achou) && (i < proxima)) {
    if (num.equals(contas[i].getNumero()))
      achou = true;
    else i = i + 1;
  }
  Conta resultado = null;
  if (achou)
     resultado = contas[i];
  return resultado;
}
```

Percorre array até encontrar conta ou chegar ao fim do array

Se achou, retorna uma conta

Removendo uma Conta

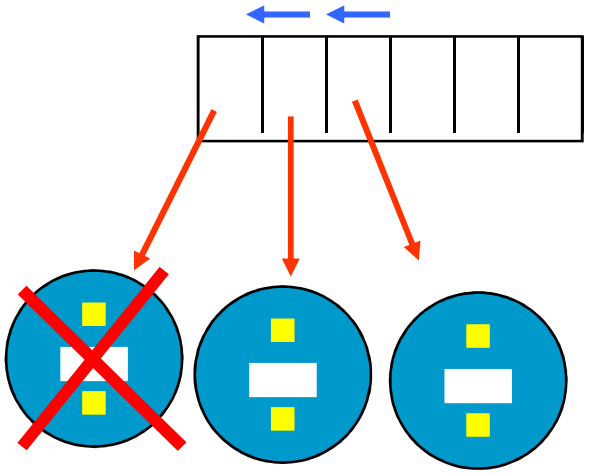
```
void remover (String num){
   int i = 0; boolean achou = false;
   while ((!achou) && (i<proxima)) {
     if(num.equals(contas[i].getNumero()))
       achou = true;
     else i = i + 1;
   }
   ...
```

Compara número de cada conta com o argumento do método

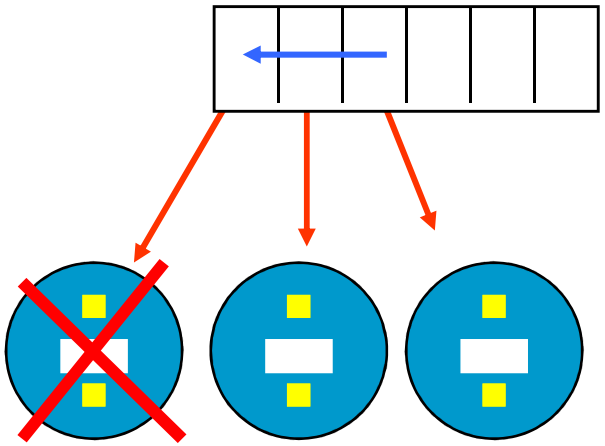
Agora, como podemos remover?

Opções para Remover

1) Remover o objeto, trazendo todos os objetos para uma posição anterior



2) Remover o objeto, trazendo o último objeto do array para a posição indicada



Primeira Opção para Remoção

```
void remover (String num){
  int i = 0; boolean achou = false;
  while ((!achou) && (i<proxima)) {
    if(num.equals(contas[i].getNumero()))
      achou = true;
    else i = i + 1;
  }
  if (achou) {
    proxima = proxima - 1;
    for (int j = i; j < proxima; j++) {
        contas[j] = contas[j + 1];
    }
  }
}
```

Move-se todos os elementos posteriores ao elemento removido

Solução mais cara!

Segunda Opção para Remoção

```
void remover (String num){
  int i = 0; boolean achou = false;
  while ((!achou) && (i<proxima)) {
    if(num.equals(contas[i].getNumero()))
      achou = true;
    else i = i + 1;
  }
  if (achou) {
    contas[i] = contas[proxima-1];
    contas[proxima-1] = null;
    proxima = proxima - 1;
  }
}
```

Move-se o último elemento para o lugar do elemento removido

Solução mais simples e eficiente!

Código Duplicado!

```
Conta procurar(String n) {...
  while ((!achou) && (i < proxima)) {
    if (n.equals(contas[i].getNumero()))
      achou = true;
    else i = i + 1;
```

```
void remover (String num){...
  while ((!achou) && (i < proxima)) {
    if(num.equals(contas[i].getNumero()))
      achou = true;
    else i = i + 1;
```

Criando Método Auxiliar para Evitar Duplicação

Método auxiliar interno

```
private int procurarIndice(String n) {
   int i = 0;
   boolean achou = false;
   while ((!achou) && (i < proxima)) {
      if (n.equals(contas[i].getNumero()))
         achou = true;
      else i = i + 1;
   }
   return i;
}
```

Retorna posição do objeto desejado

Buscando uma Conta Melhorado

```
Conta procurar(String num) {
   int indice = procurarIndice(num);
   Conta resultado = null;
   if (indice != proxima)
      resultado = contas[indice];
   return resultado;
}
```

Chamada que simplifica a procura (também remover)

Implementando o Método `existe`

```
boolean existe(String n) {
   int indice = this.procurarIndice(n);
   return (indice != proxima);
}
```

Se **indice** igual a **proxima**, é porque o elemento não existe

Duplicação de arrays

```
void duplicarColecao (){
    Conta[] contasTemp;
    contasTemp = new Conta[2 * maximo];
    for (int i = 0; i < proxima; i++) {
            contasTemp[i] = contas[i];
    }
    maximo = maximo * 2;
    contas = contasTemp;
}
```

Cria-se um array temporário com o dobro da capacidade!

Copia todas as referências para um array temporário (duplicado)

O array temporário se torna o array principal

Métodos Retornando Múltiplos Objetos

- Até agora só vimos métodos que retornam um elemento (tipo primitivo ou referência)
- No entanto, alguns métodos podem retornar mais de um elemento de um mesmo tipo

Exemplo

```
public ? retornarContasVIP () {…}
```

Método de CadastroContas que retorna todas as contas com saldo maior que 1000 reais

Várias referências para objetos Conta podem ser retornadas!

Métodos Retornando Múltiplos Objetos

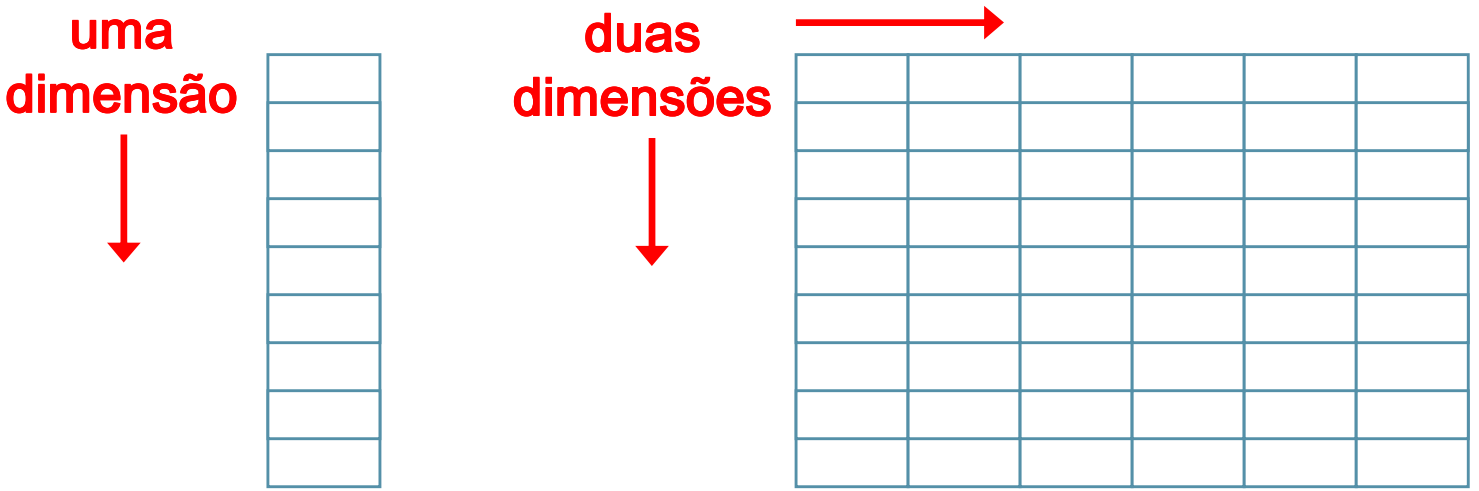
Tipo do array retornado

```
Conta[] retornarContasVIP() {
    Conta[] resultado = new Conta[proxima];
    int posicao = 0;
    for (int i = 0; i<proxima; i=i+1){
      if (contas[i].getSaldo() > 1000){
         resultado[posicao] = contas[i];
         posicao = posicao + 1;
      }
    }
    return resultado;
}
```

Armazena todas as contas VIP em resultado

Arrays Bidimensionais

- Um array unidimensional armazena uma lista de elementos
- Um array bidimensional pode ser visto como uma matriz com linhas e colunas
- Em Java, um array bidimensional é um array de arrays



Arrays Bidimensionais

```
int[][] matriz;
```

Declaração não especifica dimensões

```
int[][] matriz = new int[10][5];
for (int i=0; i<10; i++)
  for (int j=0; j<5; j++)
    matriz[i][j] = 100;
```

Cria e inicializa um array bidimensional

```
long[][] x = {{0,1},{2,3},{4,5}};
```

x[0][0]

x[0][1]

x[2][0]

Cria um array de 3 por 2

Resumindo ...

- Utilização de **arrays** para implementar coleções
  - Conceito de coleção
  - Coleção como Tipo Abstrato de Dados
  - Operações básicas de coleção
    - Inserção
    - Busca
    - Remoção
    - Existe
  - Duplicando capacidade de arrays
- Métodos retornando muitos objetos
- Arrays bidimensionais