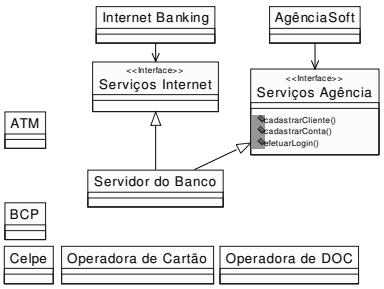
Introdução à Programação
Orientada a Objetos com Java

Arrays

Paulo Borba e Tiago Massoni
Centro de Informática
Universidade Federal de Pernambuco

Como as entidades da aplicação bancária interagem?



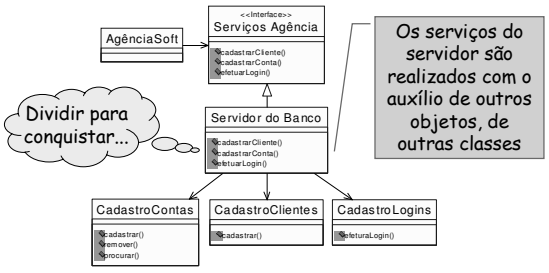
A interface `ServicosAgencia`

```
interface ServicosAgencia {
  void cadastrarCliente(Cliente c)
    throws ClienteExistenteException,
           ClienteInvalidoExceprion;

  String cadastrarConta(Conta c)
    throws ContaInvalidaException;...
}
```

Primeiro implementamos os tipos dos parâmetros dos serviços, agora vamos implementar os serviços!

Como `ServidorBanco` realiza seus serviços?



Dividir para conquistar, mas com coesão

Cada classe...

- oferece serviços logicamente relacionados
- representa um único conceito claro e bem definido

O servidor representa a "fachada" do sistema

Implementando a fachada

```
class ServidorBanco
  implements ServicosAgencia {

  private CadastroContas contas;
  private CadastroClientes clientes;
  private CadastroLogins logins;

  ServidorBanco () {
    contas = new CadastroContas();
    clientes = new CadastroClientes();
    logins = new CadastroLogins();
  }
```

Indica que esta classe satisfaz os requisitos da interface mencionada

`implements` é uma palavra reservada

Implementando os métodos da fachada

```
void cadastrarCliente(Cliente c) {
  clientes.cadastrar(c);
}

String cadastrarConta(Conta c) {
  contas.cadastrar(c);
}
```

A fachada delega os seus serviços aos seus ajudantes

Pode levantar menos exceções do que o especificado na interface, não mais do que o especificado

Só veremos exceções mais adiante

Armazenando objetos

- Para armazenar um objeto armazenamos a sua referência em um atributo ou variável do mesmo tipo

```
Conta conta;
```

- Mas para armazenar muitos objetos precisamos de muitos atributos ou variáveis do mesmo tipo

```
Conta conta1;
Conta conta2;
...
```

Implementando Cadastros

```
class CadastroContas {
  private Conta conta1;
  private Conta conta2;
```

Definição fixa da quantidade de contas armazenadas

Implementando Cadastros

```
void cadastrar(Conta conta) {
    ...
    if (conta1 == null)
      conta1 = conta;
    else{
      if (conta2 == null)
         conta2 = conta;
      else {...Espaço insuficiente...}
    }
}
```

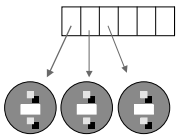
Problema 1: Complexidade de Testes

Problema 2: Dificuldade em lidar com mudanças no número de contas armazenadas

Melhorando o Programa

Precisamos de alguma estrutura de armazenamento que

- armazene várias referências para objetos de um determinado tipo
- possibilita que as referências possam ser acessadas de forma simples



Arrays

- São tipos especiais de Java
- Definem estruturas que armazenam objetos de um determinado tipo
- Uma variável do tipo array é declarada usando a notação

```
Tipo[] arrayTipo;
```

```
class CadastroContas {
  private Conta [] contas ;
```

tipo das referências armazenadas

nome do array

Opção não recomendada!

```
Conta contas[];
```

Criação de arrays

```
public class CadastroContas{
    private Conta [] contas;

    CadastroContas(){
      contas = new Conta[6];
    }
}
```

Primeiro item: contas [0]
Último item: contas [5]

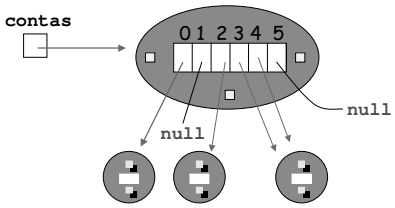
```
int [] inteiros = new int[100];
```

Arrays como variáveis locais

Criação de arrays

- O Operador **`new X[tamanho]`** cria um objeto array, não os objetos do tipo **`X`** por ele referenciado
- Arrays têm tamanho fixo depois de criados
- O comprimento do array é acessível pela variável de instância (atributo) final e pública **`length`**

Um array na memória



Opções de inicialização de arrays

Inicializadores de arrays são representados da seguinte forma: `{<expressões>}`, onde *`expressões`* representam expressões de tipos válidos separadas por vírgulas

- Exemplo: Declara, cria e inicializa um array de inteiros

```
int[] inteiros = {10,10,20,30};
```

inteiros → | 10 | 10 | 20 | 30 |

Arrays de tipos primitivos armazenam valores!

Acesso a elementos de um array

```
void cadastrar(Conta conta) {
   ...
   int local = 0;
   while((local<6)&&(contas[local]!=null)){
     local = local + 1;
   }
   if (local<6) contas[local] = conta;
}
```

Leituras e escritas nas referências armazenadas variavel[posição]

Melhorando o programa

```
CadastroContas(){
  contas = new Conta[20];
}

void cadastrar(Conta conta) {
   ...
   int local = 0;
   while ((local<6)&&(contas[local] != null)){
     local = local + 1;
   }
   if (local<6) contas[local] =
}
```

Usuário deveria configurar o tamanho inicial

Laço complexo poderia ser evitado

Melhorando o programa

```
public class CadastroContas {
  private Conta [] contas;
  private int proxima;

  CadastroContas(int tamanho) {
    contas = new Conta[tamanho];
    proxima = 0;
  }

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

Construtor recebe tamanho

Simplificação do cadastrar

Variável deve ser atualizada

Procurando 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;
}
```

Removendo uma conta

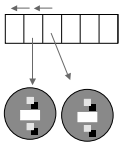
```
void remover (String num){
  int i = 0; boolean achou = false;
  while ((!achou) && (i<proxima)) {
    if(num.equals(contas[i].getNumero()))
      achou = true;
    else i = i + 1;
  }
  ...
```

Compara número de cada conta com o argumento do método

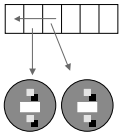
Agora, como podemos remover?

Opções para remover

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



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



Opção mais simples para remover

```
 if (achou) {
    contas[i] = contas[proxima-1];
    contas[proxima-1] = null;
    proxima = proxima - 1;
  } else {...Mensagem de erro...}

} //fim do metodo remover
```

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;
```

procurarIndice

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

Procurando uma conta

```
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);
}
```

Resumindo...

- Para armazenar objetos, podemos utilizar arrays
- Arrays são como objetos, e devem ser criados
- As posições de um array tem um tipo definido na sua declaração
- Acesso aos arrays são feitos através do número de sua posição
- Para programar temos que dividir para conquistar

Introdução à Programação

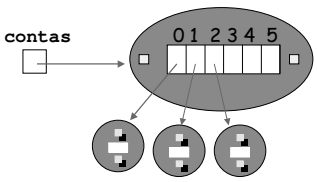
Orientada a Objetos com Java

Arrays 2

Paulo Borba e Tiago Massoni
Centro de Informática
Universidade Federal de Pernambuco

Atualização de objetos em arrays

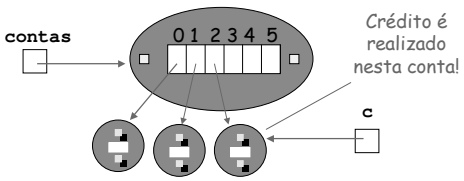
- Array pode armazenar referências para objetos
- Exemplo: Array de contas bancárias



Atualização de objetos em arrays

- *Aliasing* garante modificação dos objetos do array

```
Conta c = contas[2];
c.creditar(20);
```



Implementando os métodos `creditar` e `debitar`

```
void debitar(String numero, double valor) {
  Conta c = this.procurar(numero);
  if (c != null) c.debitar(valor);
  else {...Conta inexistente...}
}
void creditar(String numero, double valor) {
  Conta c = this.procurar(numero);
  if (c != null) c.creditar(valor);
  else {...Conta inexistente...}
}
```

Não precisamos atualizar a conta no array depois

Acesso inválido

- Se é feito acesso a um elemento indefinido de um array, é gerada uma exceção
  - **IndexOutOfBoundsException**

```
String nomes[] = {"José", "João", "Maria"};
console.println(nomes[5]);
```

Gera um erro em tempo de execução

O array pode estourar...

```
CadastroContas cadContas =
     new CadastroContas(20);
...
cadContas.cadastrar(c);
```

O tamanho do array é definido na criação

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

O que acontece quando o array estiver cheio?

(proxima = 20)

Duplicação de arrays

```
public class CadastroContas {
  private Conta [] contas;
  private int proxima;
  private int maximo;

  CadastroContas(int tamanho) {
    contas = new Conta[tamanho];
    proxima = 0;
    maximo = tamanho;
  }
```

Atributo para guardar o limite do array

Duplicação de arrays

Atingiu o limite!

```
void cadastrar (Conta c){
   if (proxima == maximo){
     Conta[]contasTemp = new Conta[maximo * 2];
     for (int i=0;i < proxima; i++)
          contasTemp[i] = contas[i];
     maximo = maximo * 2;
     contas = contasTemp;
   }
   contas[proxima] = c;
   proxima = proxima + 1;
}
```

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 multidimensionais

```
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}};
```

Cria um array de 3 por 2

x[0][0] x[0][1] x[2][0]

Resumindo...

- Arrays que armazenam referências podem ter seus elementos atualizados com aliasing
- Acessos a posições não existentes de um array provocam uma exceção
- Duplicação de arrays é uma solução para o problema de gerenciamento de espaço
- Métodos podem retornar objetos do tipo array
- Arrays multimensionais são representandos como arrays de objetos do tipo array