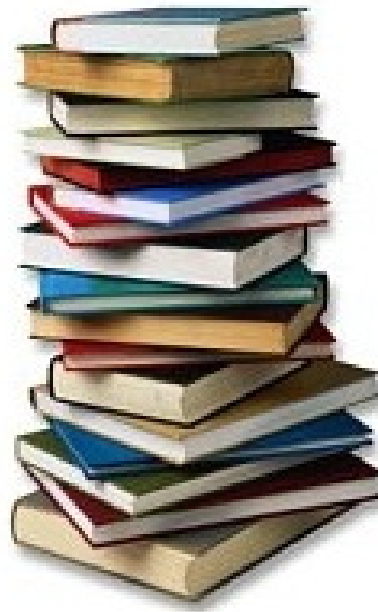
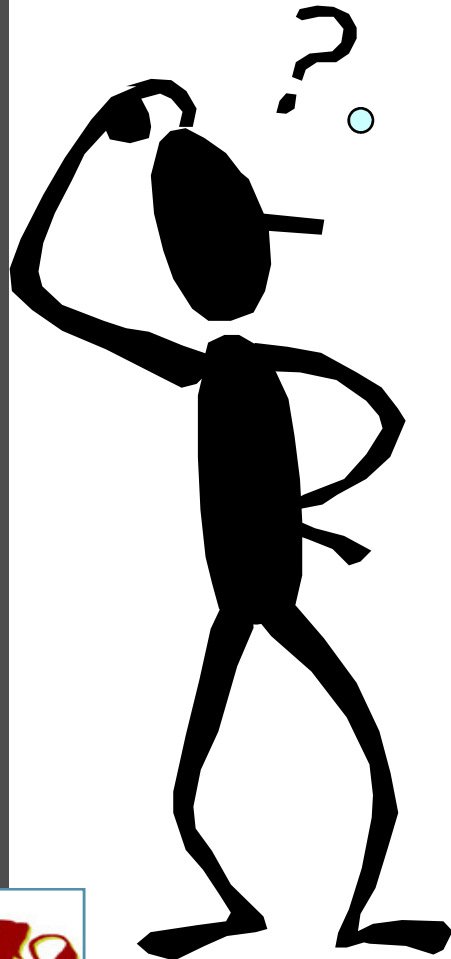


Introdução à Programação



*Armazenamento de Grande
Quantidade de Informação –
Usando Arrays*

Armazenando Grande Quantidade de Informação



Arrays !

Tópicos da Aula

- ◆ Hoje, aprenderemos como armazenar grande quantidade de informação
 - Necessidade de armazenamento
 - Utilização de vários atributos
 - Manipulação incremental das informações
- ◆ Aprenderemos a utilizar **arrays**
 - Necessidade de arrays
 - Conceito de array
 - Criação
 - Array de Referências x Array de Tipos Primitivos
 - Inicialização
 - Acesso
 - Limites de um array
 - Passagem de arrays como parâmetros

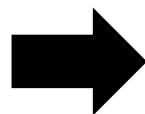
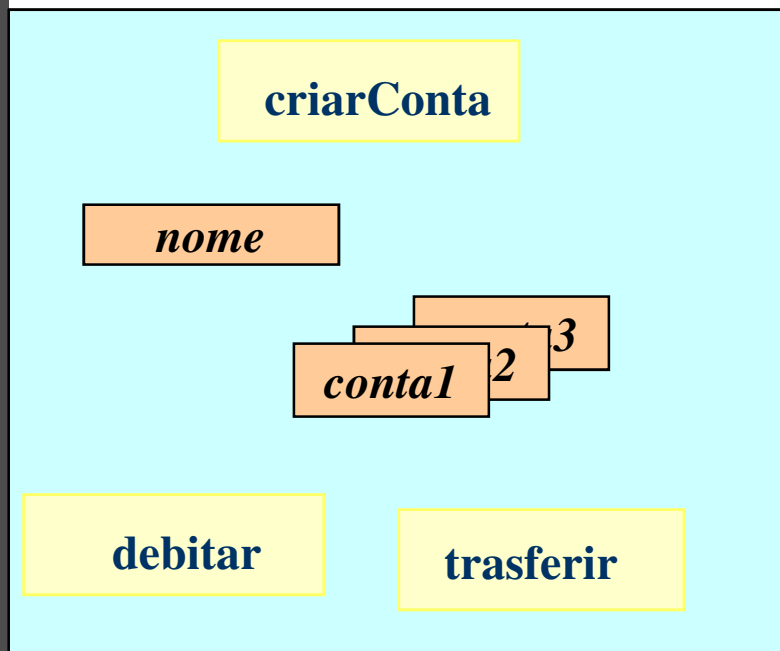
Necessidade de Armazenamento

- ◆ Sistemas armazenam informações para que possam posteriormente acessá-las e manipulá-las
- ◆ Maioria dos sistemas requer o armazenamento de grande quantidade de objetos do mesmo tipo
 - Mesmas operações para manipular estes objetos
 - Ex: processamento de imagens
 - Milhões de imagens e cada imagem composta de milhões de pixels
- ◆ No desenvolvimento de um programa, devemos definir estruturas e mecanismos para armazenar e manipular esta grande quantidade de informação

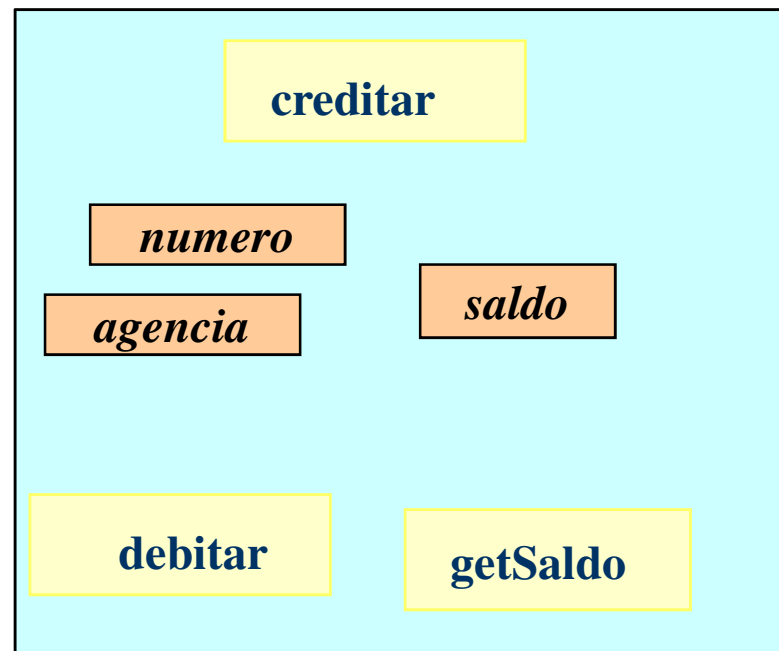
Classe Banco

- ❖ Objeto da Classe Banco deve armazenar muitos objetos da classe Conta

Classe Banco



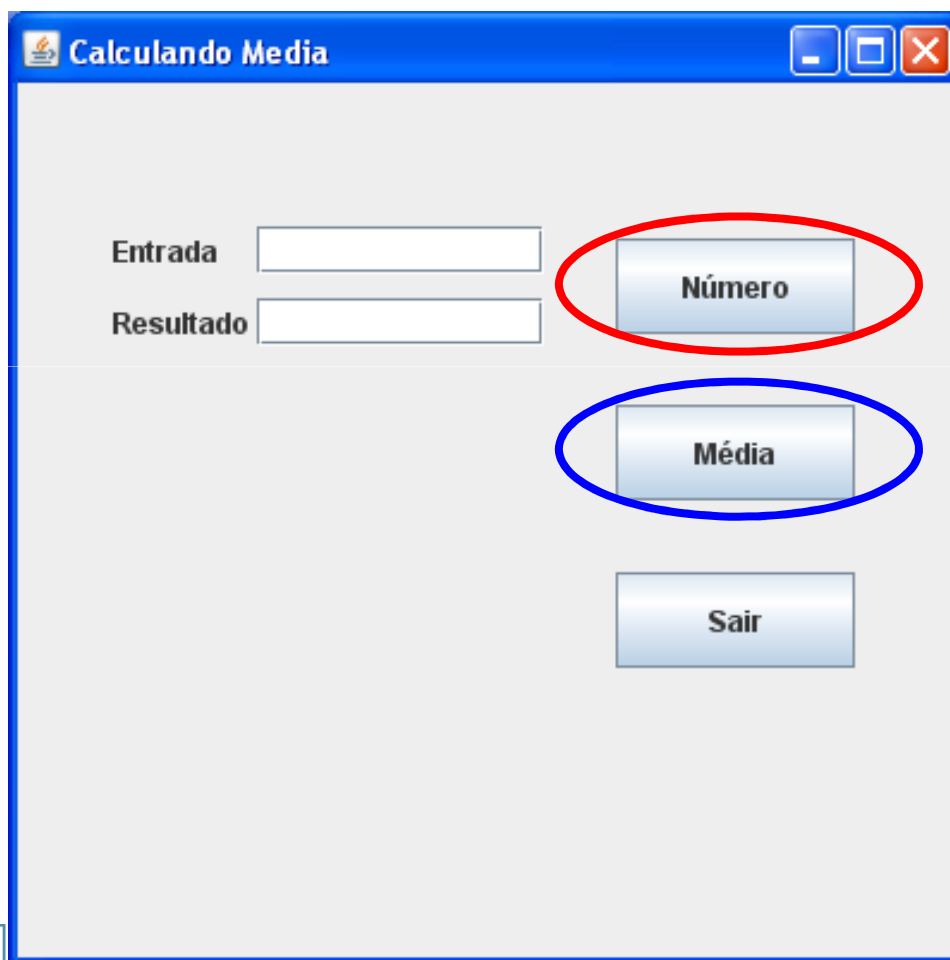
Classe Conta



Como Armazenar Grande Quantidade de Informações?

- ◆ Até agora, vimos construções de programação que permitem guardar uma única informação de cada vez
 - Variável e atributo
- ◆ Neste caso, a solução seria criar um atributo para cada objeto que desejamos armazenar
 - Não é uma boa solução, porém hoje veremos como podemos trabalhar com esta alternativa

Exemplo: Calculando Média Aritmética



Calculando Media

Entrada

Resultado

Número

Média

Sair

Depois de cada entrada de número, botão deve ser clicado

Para obter média, botão deve ser clicado

Tentativa com Vários Atributos

```
public class JanelaComArmazenamento extends Window {  
    Button botaoCalcularMedia;  
    Button botaoNovoNumero;  
    Button botaoSaida;  
    ...  
    int numero1;  
    int numero2;  
    boolean vezImpar;  
    public void clickEvent(Component c) {  
        if (c == botaoCalcularMedia) {  
            acaoBotaoCalcularMedia();  
        } else {  
            if (c == botaoSaida) {  
                fecharPrograma();  
            } else { if (c == botaoNovoNumero) {  
                acaoBotaoNovoNumero();  
            }  
        }  
    }  
}
```

**2 atributos para
armazenar números**

**Atributo auxiliar para
informar se foi entrada o
primeiro ou segundo número**

**Limitação: Só permite média
de 2 números**

Tentativa com Vários Atributos

```
private void acaoBotaoNovoNumero() {  
    int entrada;  
    entrada = campoEntrada.getInt();  
    if (vezImpar == true) {  
        numero1 = entrada;  
        vezImpar = false;  
    } else {  
        numero2 = entrada;  
        vezImpar = true;  
    }  
    campoEntrada.setText("");  
}  
  
private void acaoBotaoCalcularMedia() {  
    int media;  
    media = (numero1 + numero2)/2;  
    campoResultado.setInt(media);  
}
```

Permite alternância
entre os locais de
armazenamento do
número entrado

Calcula mesmo sem
números terem sido
digitados

Aumentando Número de Atributos

```
public class JanelaComArmazenamento3 extends Window {
```

```
..
```

```
int numero1;
```

```
int numero2;
```

```
int numero3;
```

**3 atributos para
armazenar números**

```
int proximo;
```

**Indica o próximo número a
ser entrada: primeiro,
segundo ou terceiro**

```
public void clickEvent(Component c) {  
    if (c == botaoCalcularMedia) {  
        acaoBotaoCalcularMedia();
```

```
    } else {
```

```
        if (c == botaoSaida) {  
            fecharPrograma();
```

```
        } else { if (c == botaoNovoNumero) {  
            acaoBotaoNovoNumero();
```

```
        }
```

```
    }
```

```
}
```

**Limitação: Só permite média
de 3 números**

Aumentando Número de Atributos

```
private void acaoBotaoNovoNumero() {  
    int entrada = campoEntrada.getInt();  
    if (proxima == 0) {  
        numero1 = entrada;  
        proxima = proxima + 1;  
    } else if (proxima == 1) {  
        numero2 = entrada;  
        proxima = proxima + 1;  
    } else if (proxima == 2) {  
        numero3 = entrada;  
        proxima = 0;  
    }  
    campoEntrada.setText("");  
}  
  
private void acaoBotaoCalcularMedia() {  
    int media;  
    media = (numero1 + numero2 + numero3) / 3;  
    campoResultado.setInt(media);  
}
```

**Permite alternância
entre os locais de
armazenamento do
número entrado**

Problemas com Uso de Vários Atributos

- ◆ Dificuldade de lidar com mudança do número de informações que devem ser armazenados
 - Acrescentar ou remover atributo
 - Muitas vezes requer modificações em vários métodos e/ou construtores
- ◆ Complexidade de testes
 - Pode requerer testes para saber em qual atributo devemos armazenar a informação

Outra Estratégia: Manipulação Incremental da Informação

```
public class JanelaComArmazenamento3 extends Window {
```

```
    .  
    .  
    int somatorio;
```

**Atributos para armazenar
somatório dos números**

```
    int quantidadeDeNumeros;
```

**Indica a quantidade de
números entrados**

```
    public void clickEvent(Component c) {  
        if (c == botaoCalcularMedia) {  
            acaoBotaoCalcularMedia();  
        } else {  
            if (c == botaoSaida) {  
                fecharPrograma();  
            } else { if (c == botaoNovoNumero) {  
                acaoBotaoNovoNumero();  
            }  
        }  
    }  
}
```

**Permite média de quantidade
variada de números**

Outra Estratégia: Manipulação Incremental da Informação

```
private void acaoBotaoNovoNumero() {  
    int entrada;  
    entrada = campoEntrada.getInt();  
    somatorio = somatorio + entrada;  
    quantidadeDeNumeros = quantidadeDeNumeros + 1;  
    campoEntrada.setText("");  
}  
  
private void acaoBotaoCalcularMedia() {  
    int media;  
    if (quantidadeDeNumeros == 0)  
        media = 0;  
    else  
        media = somatorio/quantidadeDeNumeros;  
    campoResultado.setInt(media);  
}
```

Lógica simplificada

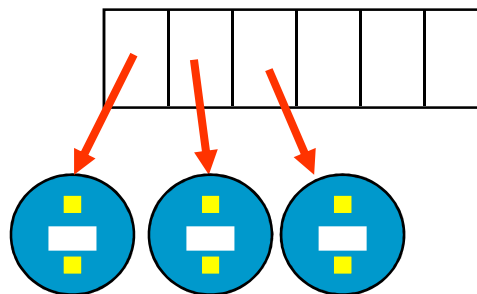
Contudo perdemos informação sobre cada número digitado!

Problemas com Manipulação Incremental de Informação

- ◆ Perda de informação sobre os valores individuais que foram armazenados
 - No exemplo, sabemos apenas qual foi o somatório dos valores inseridos e não quais valores foram inseridos
- ◆ Limita a utilização da informação armazenada
 - No exemplo, se quiséssemos utilizar os mesmos dados para uma operação diferente (ex: multiplicação), não conseguiríamos

Necessidade de Arrays

- ◆ Precisamos de alguma estrutura de armazenamento que:
 - armazene várias referências para objetos (ou valores) de um determinado tipo
 - permita que as referências (ou valores) sejam acessadas de forma simples



Arrays !

Arrays

- ◆ São tipos especiais de Java
- ◆ Definem estruturas que armazenam referências para objetos ou valores primitivos de um determinado tipo
- ◆ Uma variável do tipo array é declarada usando
`Tipo[] nomeArray;`
- ◆ Variável guarda uma referência para um objeto do tipo array

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

tipo das(os) referências ou
valores armazenadas (os)

nome do array

Criação de Arrays

```
public class CadastroContas{  
    private Conta [] contas;  
  
    CadastroContas() {  
        contas = new Conta[20];  
    }  
  
    double calcularMedia() {  
        double [] saldos;  
        saldos = new double[20];  
        ...  
    }  
}
```

Arrays são
objetos, por isso
devem ser
instanciados

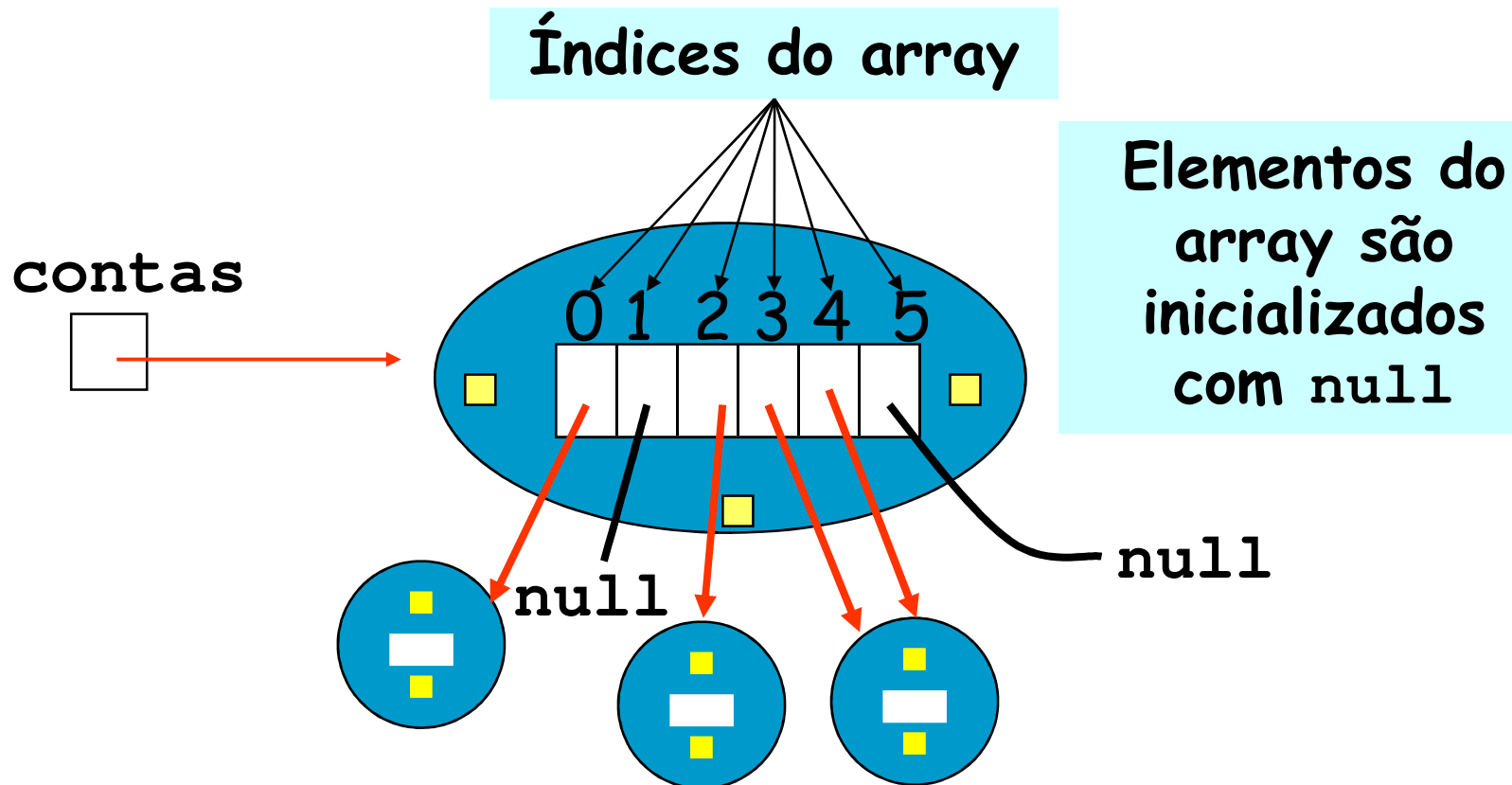
Arrays como
variáveis locais

Criação de Arrays

- ❖ O operador `new X[tamanho]` cria um objeto array, não os objetos do tipo `X` por ele referenciado
- ❖ No ato da criação, os elementos do array são inicializados com valores padrões
- ❖ No ato da criação, devemos especificar o tamanho do array
 - Arrays têm tamanho fixo depois de criados
- ❖ O comprimento do array é acessível pela variável de instância (atributo) final e pública `length`

`contas.length`

Array de Referências para Objetos

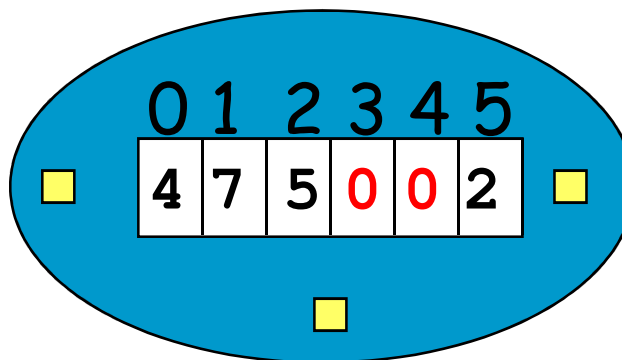


- ◆ Elementos de arrays são indexados
- ◆ Índice começa em 0 e vai até *tamanho - 1*
- ◆ No ato de criação do array, **NÃO** são criados objetos do tipo Conta

Array de Valores Primitivos

```
int [] inteiros = new int[6] ;
```

inteiros



Arrays de tipos
primitivos
armazenam valores!

Elementos do
array são
inicializados
com valor
padrão

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
----	----	----	----

Opções de Inicialização de Arrays

- ◆ No caso de utilizar inicializadores de arrays, note que:
 - O operador **new** não é utilizado
 - O tamanho do array não é especificado
- ◆ O tamanho do array é determinado pelo número de elementos contidos na lista de inicialização
- ◆ Este tipo de inicialização só pode ser utilizado na declaração de arrays

Acessando Elementos de Arrays

```
void cadastrar(Conta conta) {  
    ...  
    int local = 0;  
    while((local<20) && (contas[local] != null)) {  
        local = local + 1;  
    }  
    if (local<20) contas[local] = conta;  
}
```

Leituras e
escritas nas
referências
armazenadas
variável[índice]

Acessando Elementos de Arrays

```
public class CadastroContas{  
    private Conta [] contas;  
  
    CadastroContas() {  
        contas = new Conta[20];  
    }  
  
    void criarConta(String numero) {  
        ...  
        int local = 0;  
        while( (local<20) && (contas[local] != null) ) {  
            local = local + 1;  
        }  
        if (local<20)  
            contas[local] = new Conta(numero);  
    }  
}
```

Criação de array
de Conta

Criação de objeto
Conta e
armazenamento da
referência no
[índice]

Checando os Limites dos Arrays

- ◆ Uma vez criados, arrays tem tamanho fixo
- ◆ Ao utilizar o índice para referenciar um elemento do array, este índice deve permitir o acesso a um elemento válido
 - Índice deve variar de *0* a *tamanho - 1*
- ◆ O interpretador Java lança uma `ArrayIndexOutOfBoundsException` se o índice utilizado for fora do limite do array

Checando os Limites dos Arrays

```
public class CadastroContas{  
    private Conta [] contas;  
  
    CadastroContas() {  
        contas = new Conta[20];  
    }  
  
    double calcularMedia() {  
        double [] saldos;  
        saldos = new double[20];  
        ...  
    }  
}
```

Primeiro item:
contas [0]
Último item:
contas [19]

Primeiro item:
saldos [0]
Último item:
saldos [19]

Checando os Limites dos Arrays

- ◆ São comuns, erros de programação no uso de arrays dentro de laços
 - Deve-se prestar atenção na parte de teste do laço

```
double calcularMedia() {  
    double [] saldos = new double[20];  
    double media = 0;  
    for (int i = 0; i <= 20, i ++)  
        saldos[i] = contas[i].getSaldo();  
    media = media + saldos[i];  
    ...  
}
```

Teste deveria ser $i < 20$

Por causa do teste errado,
esta linha gerará uma
exceção

Passando Arrays como Parâmetros

- ◆ Um array inteiro pode ser passado como parâmetro para um método
 - Parâmetro do método deve ser do tipo `Tipo[]`
- ◆ Assim como qualquer objeto, ao passar a referência (argumento) para o método podemos modificar este array dentro do método
 - Modificar um elemento do array ou incluir um novo elemento
- ◆ Podemos passar também um elemento em particular de um array para um método
 - Parâmetro deve ser do tipo `Tipo`

Passando Arrays como Parâmetros

```
public class Matematica{  
  
    public void dobrarValores(int[] valores) {  
        for (int i = 0; i < valores.length;i++)  
            valores[i] = valores[i] * 2;  
    }  
  
    public int dobrarValor(int valor) {  
        return (valor*2);  
    }  
}
```

```
Matematica mat = new Matematica();  
int[] inteiros = {4,5,6,7};  
mat.dobrarValores(inteiros);  
inteiros[2] = mat.dobrarValor(inteiros[2]);
```

inteiros agora vai
ter {8,10,12,14}

inteiros agora vai
ter {8,10,24,14}

Passando Arrays como Parâmetros

```
public class ManipulaContas{  
  
    public void dobrarSaldos(Conta[] contas) {  
        for (int i = 0; i < contas.length &&  
                contas[i] != null; i++)  
            contas[i].creditar(contas[i].getSaldo());  
    }  
  
    public void dobrarSaldo(Conta conta) {  
        conta.creditar(conta.getSaldo());  
    }  
}
```

```
ManipulaContas cad = new ManipulaContas();  
Conta[] contas = new Conta[6];  
contas[0] = new Conta();  
contas[0].creditar(1000); ...  
cad.dobrarSaldos(contas);  
cad.dobrarSaldo(contas[0]);
```

Todos os elementos
de contas vão ter
o saldo dobrado

contas[0] vai
agora ter saldo igual
a 4000

Voltando a Classe Cadastro de Contas

```
public class CadastroContas{  
    private Conta [] contas;  
  
    CadastroContas () {  
        contas = new Conta[20];  
    }  
  
    void cadastrar(Conta conta) {  
        ...  
        int local = 0;  
        while ((local<20)&&(contas[local] != null)){  
            local = local + 1;  
        }  
        if (local<20) contas[local] = conta;  
    }  
}
```

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

Resumindo...

- ◆ Armazenamento de grande quantidade de informações
 - Necessidade de armazenamento
 - Utilização de vários atributos
 - Manipulação incremental das informações
- ◆ Conceitos básicos de **arrays**
 - Necessidade de arrays
 - Conceito de array
 - Algumas operações básicas
 - Array de Referências x Array de Tipos Primitivos
 - Importância de checar os limites de um array
 - Passagem de arrays como parâmetros
 - Exemplos de utilização de array