

JAVA

Fábio Pereira Botelho

M.Sc. Ciência da Computação UFPE
fabio.botelho@embrapa.br

Anápolis – GO, Agosto de 2024

Objetivo

- Apresentar as características de JAVA
- Focar e aprofundar aspectos da linguagem de programação JAVA

Apresentação

Java é:

- Uma linguagem de Programação Orientada a Objetos
 - ✓ Desenvolvida pela SUN Microsystems
 - Independente de Plataforma (Portável)
 - ✓ Um único código fonte que não precisa ser reescrito para cada nova plataforma ou sistema operacional
 - ✓ A Máquina Virtual JAVA (sigla do inglês JVM) está disponível na maioria das plataformas e sistemas operacionais
 - ✓ Após compilado, os arquivos fonte (*.java), geram os byte-code (*.class) que podem ser executados em qualquer outra plataforma ou sistema operacional que possua a mesma versão da JVM
 - Seguro
 - ✓ Aspecto segurança levado em conta desde o início da elaboração de JAVA
-
-

Apresentação

Java é:

- Robusto
 - ✓ O código java é confiável e facilita a criação de software de alta qualidade
 - ✓ Java elimina muitos dos problemas de memória comuns em linguagens como C e C++
 - ✓ Não aceita o acesso direto aos ponteiros de memória
 - ✓ A JVM realiza verificações periódicas durante a execução de um programa para se certificar que todas as referências de arrays e strings estão corretas
 - ✓ A JVM realiza coleta automática de lixo, i.e, memória alocada não mais usada
 - ✓ Evita erros de software comuns em outras linguagens relativos à não liberação de memória que deveria ser liberada ou a sua liberação mais de uma vez
-
-

Apresentação

Java é:

- Robusto
 - ✓ Java é fortemente tipado, i.e, exige a declaração de tipos de dados e exige declarações explícitas de métodos, o que reduz a possibilidade de erros
 - ✓ Java institui uma metodologia de detecção de erros conhecida como tratamento de exceções
 - Quando acontece um erro em um programa, JAVA o sinaliza como uma exceção que pode ser capturada e tratada sem que o programa “aborte”
 - Fácil de Aprender
 - ✓ Mais fácil de aprender que C/ C++
 - ✓ Incorpora os conceitos básicos de programação orientada a objetos e elimina construções complicadas de outras linguagens como é o caso de herança múltipla
 - ✓ Muitas palavras chave de JAVA estão presentes em outras linguagens
-
-

Definições de JAVA

- Classe

- ✓ Já vimos que uma classe é uma estrutura de dados de programação em que se agrupa dados referentes a um conceito abstraído do mundo real com os métodos que operam sobre os dados

- ✓ Definição de uma classe JAVA:

 modificador class Nome-da-Classe {

// As instruções da classe, i.e, variáveis (propriedades ou atributos) e funções (métodos)

}

Onde:

- Modificador representa o modificador da classe

- Nome-da-Classe representa o nome que se atribui à classe definida

Se omitido,
subentende-se
que a classe não
é pública

Definições de JAVA

- Classe

```
public class Nome-da-Classe {  
    // As instruções da classe, i.e, variáveis (propriedades ou atributos) e  
    funções (métodos)  
  
}
```

Onde:

→ Modificador `public` torna a classe disponível fora de uma “biblioteca de funções” conhecida em java como pacote, ou seja, disponível para todos

Definições de JAVA

- Classe

```
public abstract class Nome-da-Classe {  
    // As instruções da classe, i.e, variáveis (propriedades ou atributos) e  
    funções (métodos)  
  
}
```

Onde:

→ Modificador `abstract` torna a classe abstrata, ou seja, não pode ser instanciada para a criação de objetos e deve necessariamente ser herdada por outras classes que poderão ser abstratas ou não. Pode vir acompanhada do modificador `public` ou não.

Definições de JAVA

- Classe

```
public final class Nome-da-Classe {  
    // As instruções da classe, i.e, variáveis (propriedades ou atributos) e  
    funções (métodos)  
  
}
```

Onde:

→ final define que a classe não pode mais ser herdada por nenhuma outra classe. Na tentativa de se herdar (palavra chave extends) a partir de uma classe definida como final o compilador acusará erro. Da mesma forma, pode vir acompanhada da palavra reservada public ou não.

Definições de JAVA

- Método

- ✓ Como vimos, pode ser entendido como uma função que pertence a uma classe

- ✓ Definição

```
modificador tipo nome(parâmetros) {  
    // implementação do método
```

```
}
```

- ✓ Onde:

- modificador, pode ser formada por diferentes palavras reservadas: public, private, protected, static, final e abstract

- tipo, representa o valor que o método retorna para quem o chamou

- parâmetros são argumentos passados para o método. São separados por vírgula e cada nome de argumento deve vir precedido de seu tipo

- implementação define as instruções que formam o método

Definições de JAVA

- Método Retorna um valor inteiro ao método
✓ E.g. que o chamou

Recebe como parâmetros três valores inteiros

Modificador de acesso ao método

```
public int soma( int a, int b, int c) {  
    int resultado = a + b + c;  
    return resultado;  
}
```

- ✓ O método soma é acessível (visível) de qualquer lugar em que a classe que o contém também seja visível. Ele retorna um valor inteiro e recebe como parâmetros três valores também inteiros.

Definições de JAVA

- Método

- ✓ E.g.

Modificador de
acesso ao método

```
private int soma( int a, int b, int c) {  
    int resultado = a + b + c;  
    return resultado;  
  
}
```

- ✓ O método soma é acessível (visível) apenas dentro da classe que o contém. É acessível assim somente a partir de outros métodos dentro da classe.

Definições de JAVA

- Método

- ✓ E.g.

Modificador de
acesso ao método

```
protected int soma( int a, int b, int c) {  
    int resultado = a + b + c;  
    return resultado;  
}
```

- ✓ O método soma é acessível (visível) apenas na sua classe (que o contém), nas subclasses (que herdam da classe que o contém) ou no pacote ao qual a classe pertence (semelhante ao conceito de biblioteca de funções)

Definições de JAVA

- Método

- ✓ E.g.

Modificador de
acesso ao método

```
static int soma( int a, int b, int c) {  
    int resultado = a + b + c;  
    return resultado;  
}
```

- ✓ O método soma pode ser acessado a partir da classe sem que um objeto precise ser instanciado para que o método possa ser acessível a partir do objeto. Métodos estáticos somente podem chamar e serem chamados a partir de outros métodos também definidos como estáticos.

Definições de JAVA

- Método

- ✓ E.g.

Modificador de
acesso ao método

```
final int soma( int a, int b, int c) {  
    int resultado = a + b + c;  
    return resultado;  
  
}
```

- ✓ Agora, o método soma não pode ser sobreposto por outra classe que o herde. I.e outra classe que estenda a classe onde o método soma está definido não poderá criar um método soma que retorne um inteiro e passe três parâmetros inteiros

Definições de JAVA

- Variáveis, Atributos, Propriedades ou Campos
- Definição:

modificador tipo nome-da-variável;

ou

modificador tipo nome-da-variável = valor;

✓ Onde:

- modificador pode ser public, private, protected, static, final, transient, volatile
 - tipo representa o tipo da variável definida, e.g, int para inteiro; double para valores fracionários; String para valores que representam sequência de caracteres; char para valores que representam apenas um caractere (demais tipos serão estudados adiante)
 - Nome-da-variável representa o nome atribuído à variável
 - = valor, onde valor representa o valor inicial atribuído à variável
-
-

Estruturas de Controle em JAVA

- Estrutura de seleção if
- Estrutura de seleção if/else
- Estrutura de repetição while
- Estrutura de repetição for
- Estrutura de seleção múltipla switch
- Estrutura de repetição do/while
- As instruções break e continue

Operadores em JAVA

- Atribuição
- Incremento e Decremento
- Lógicos



Estruturas de Controle em JAVA

- Estrutura de seleção if

- ✓ É utilizada para escolher entre cursos de ação alternativos em um programa
- ✓ Realiza a ação indicada caso a condição dada seja true, senão não executa a ação

→ Se a nota do aluno for maior ou igual a 6
imprima “Aprovado”

← Algoritmo

→ if (notaAluno >= 6)
System.out.println(“Aprovado”);

Em JAVA

comando é executado
se notaAluno for maior
ou igual a 6

→ if (notaAluno >= 6) {
System.out.println(“-----”);
System.out.println(“Aprovado”);
System.out.println(“-----”);
}

← Bloco de comandos é
executado caso a
condição dada seja
Verdadeira

Estruturas de Controle em JAVA

- Estrutura de seleção if
- ✓ Exercício
- ✓ Elabore um programa em java que leia as notas np1, np2 e np3; calcule a nota média; imprima o resultado e diga se o aluno está aprovado (nota média ≥ 7).

Estruturas de Controle em JAVA

- Estrutura de seleção if
- ✓ Exercício
- ✓ Algoritmo

Algoritmo para cálculo da nota média dos alunos;

Leia nota 1;

Leia nota 2;

Leia nota 3;

Calcule $\text{total} = \text{nota 1} + \text{nota 2} + \text{nota 3}$;

Calcule $\text{media} = \text{total} / 3$;

Imprima media;

Se media for maior ou igual a 7, então

Imprima aprovado;

Fim Se;

Fim Algoritmo

Estruturas de Controle em JAVA

- Estrutura de seleção if

- ✓ Exercício

- ✓ Implementação

```
public class NotaMedia {  
    public static void main (String args[]) {  
        // Inicialize as variáveis nota 1, nota 2, nota 3, total e média  
        double n1, n2, n3, total, media = 0;  
        // Lê as notas 1, 2 e 3  
        System.out.println("Entre com as notas n1, n2 e n3 ! ");  
        n1 = Util.readDb1(); // Usa as funções definidas na classe  
        n2 = Util.readDb1(); // estática Util para a leitura de  
        n3 = Util.readDb1(); // valores do tipo double  
        // Calcule o total  
        total = n1 + n2 + n3;  
        // Calcule a média  
        media = total/3;  
        // Mostre a média  
        System.out.println("Nota média é " + media);  
  
        // Diga se o aluno está aprovado  
        if (media >= 7) {  
            System.out.println("Aluno está aprovado!");  
        }  
    }  
}
```

Estruturas de Controle em JAVA

- Estrutura de seleção if/else

- ✓ Permite especificar que uma ação será tomada quando a condição for Verdadeira (true) ou outra ação será tomada quando a condição dada for falsa (false)

- Se a nota do aluno for maior do que ou igual a 6

- Imprima “Aprovado”

- senão

- Imprima “Reprovado”

- if (notaAluno >= 6)

- System.out.println(“Aprovado”);

- else

- System.out.println(“Reprovado”);

- Os caracteres especiais “{“ e “}” são usados para delimitar blocos de comandos a serem executados caso a condição seja true ou caso seja false

```
if (notaAluno >= 6) {  
    System.out.println( “-----” );  
    System.out.println( “Aprovado” );  
    System.out.println( “-----” );  
}  
else {  
    System.out.println( “-----” );  
    System.out.println( “Reprovado” );  
    System.out.println( “-----” );  
}
```

Estruturas de Controle em JAVA

- Estrutura de seleção if/else
 - ✓ Outra forma de representar em java esta estrutura de controle é através do uso do operador condicional (?:)
 - ➔ Possui três operandos. O primeiro representa a condição; o segundo representa a ação que será tomada caso a condição seja verdadeira; o terceiro, a ação que será tomada caso a condição seja falsa
 - ➔ Assim:

- ♦ `System.out.println(notaAluno >= 6 ? “Aprovado” : “Reprovado”)`

condição

Resultado Verdadeiro (true)

Resultado
Falso
(false)

Estruturas de Controle em JAVA

- Estrutura de seleção if/else

- ✓ Exercício

- ✓ Dado o Algoritmo abaixo, escreva o referente código JAVA

- ✓ Se a notaAluno for maior ou igual a 9

Imprima “A”

senão

Se a notaAluno for maior ou igual a 8

Imprima “B”

senão

Se notaAluno for maior ou igual a 7

Imprima “C”

senão

Se notaAluno for maior ou igual a 6

Imprima “D”

senão

Imprima “E”

Estruturas de Controle em JAVA

- Estrutura de seleção if/else

- ✓ Resposta Exercício

- ✓ Dado o Algoritmo abaixo, escreva o referente código JAVA

- ✓ if (notaAluno >= 9)

- System.out.println("A");

- else

- if (notaAluno >= 8)

- System.out.println("B");

- else

- if (notaAluno >= 7)

- System.out.println("C");

- else

- if (notaAluno >= 6)

- System.out.println("D");

- else

- System.out.println("E");

Estruturas de Controle em JAVA

- Estrutura de seleção if/else

- ✓ Exercício

- ✓ Altere o programa JAVA dado para a estrutura de seleção if, a fim de que reflita a implementação realizada para a estrutura de seleção if/else

```
public class NotaMedia {  
    public static void main (String args[]) {  
        // Inicialize as variáveis nota 1, nota 2, nota 3, total e média  
        double n1, n2, n3, total, media = 0;  
  
        // Lê as notas 1, 2 e 3  
        System.out.println("Entre com as notas n1, n2 e n3 ! ");  
        n1 = Util.readDb1(); // Usa as funções definidas na classe  
        n2 = Util.readDb1(); // estática Util para a leitura de  
        n3 = Util.readDb1(); // valores do tipo double  
  
        // Calcula o total  
        total = n1 + n2 + n3;
```

Estruturas de Controle em JAVA

- Estrutura de seleção if/else

- ✓ Exercício

- ✓ Altere o programa JAVA dado para a estrutura de seleção if, a fim de que reflita a implementação realizada para a estrutura de seleção if/else

```
// Calcule a média
media = total/3;

// Mostre a média
System.out.println("Nota média é " + media);
// Diga se o aluno está aprovado
if (media >= 9)
    System.out.println("A");
else
    if (media >= 8)
        System.out.println("B");
    else
        if (media >= 7)
            System.out.println("C");
        else
            if (media >= 6)
                System.out.println("D");
            else
                System.out.println("E");
    }
}
```

Estruturas de Controle em JAVA

- Estrutura de repetição while

- ✓ Permite especificar que uma ação será repetida enquanto alguma condição permanecer verdadeira

Algoritmo

→ Inicialize produto = 2;
Enquanto potência de 2 for menor ou igual a 1000, execute o bloco
 Inicio bloco while
 Imprima produto;
 Calcule produto = 2 * produto;
Fim bloco while

Implementação em JAVA

→ `int produto = 2;`
`while (produto <= 1000) {`
 `System.out.println(produto);`
 `produto = produto * 2;`
`}`

Estruturas de Controle em JAVA

- Estrutura de repetição while
- ✓ Exercício

Escreva o algoritmo e em seguida a implementação em JAVA para calcular a média das notas obtidas em uma turma de 10 alunos.



Estruturas de Controle em JAVA

- Estrutura de repetição while
- ✓ Resposta do Exercício

Algoritmo

Defina o total como zero

Defina o contador de notas como um

Enquanto o contador de nota for menor ou igual a dez

 Leia próxima nota

 Adicione a nota ao total

 Adicione um ao contador de notas

Defina a média da turma como o total dividido por dez

Imprima a média da turma

Estruturas de Controle em JAVA

- Estrutura de repetição while
- ✓ Resposta do Exercício

JAVA

```
import javax.swing.JOptionPane;
public class MediaTurma {
    public static void main (String args[] ) {
        double total, // soma das notas da turma
            media, // média das notas
            valorNota; // nota lida
        int qNotas; // quantidade de notas lidas
        String nota; // nota digitada pelo usuário

        // Inicialização

        total = 0;
        qNotas = 1;
```


Estruturas de Controle em JAVA

- Estrutura de repetição while

Resposta do Exercício

JAVA

// Processamento

```
while ( qNotas <= 10 ) {  
    nota = JOptionPane.showInputDialog("Informe a nota do aluno " + qNotas + ": ");  
  
    // converte a nota de String para double  
  
    valorNota = Double.parseDouble(nota);  
  
    // atualiza total  
  
    total = total + valorNota;  
  
    qNotas = qNotas + 1;  
  
}
```

Estruturas de Controle em JAVA

- Estrutura de repetição while

Resposta do Exercício

JAVA

// calcula resultado

```
media = total / 10;
```

```
// mostra resultado
```

```
JOptionPane.showMessageDialog (  
    null, "A média da classe é " + media , "Média da Classe",  
    JOptionPane.INFORMATION_MESSAGE  
);
```

```
System.exit( 0 );
```

```
}  
}
```

Estruturas de Controle em JAVA

- Estrutura de repetição for
 - ✓ Baseia-se na repetição controlada por contador
 - ✓ Permite que um comando ou bloco de comandos seja executado, dado uma variável de controle, uma condição para continuação do laço e um incremento da variável de controle a cada iteração do laço.
 - ✓ Algoritmo

Algoritmo Imprime Numeros Inteiros de 1 a 999

Para $x = 1$, incremente x em 1 enquanto $x < 1000$

Imprime x

Fim Para

Fim Algoritmo

Estruturas de Controle em JAVA

- Estrutura de repetição for
- ✓ Implementação

```
public class Sequencial {  
    public static void main (String args[]) {  
  
        for (int x=1; x<1000; x = x + 1) {  
            System.out.println(x);  
        }  
  
    }  
  
}
```

Estruturas de Controle em JAVA

- Estrutura de repetição for
 - ✓ Implementação

```
public class Sequencial {  
    public static void main (String args[]) {
```

```
        for ( int x=1; x<1000; x = x + 1 ) {  
            System.out.println(x);  
        }
```

```
    }
```

```
}
```

Definição e inicialização da variável de controle

Incremento da variável de controle a cada iteração do laço

Condição de continuação do laço

Estruturas de Controle em JAVA

- Estrutura de seleção múltipla switch
- ✓ A estrutura if é dita de seleção única
- ✓ A estrutura if/else é dita de seleção dupla
- ✓ A estrutura de seleção múltipla switch permite tratar uma tomada de decisão em que várias fluxos de execução diferentes ocorrerão de acordo com o valor associado à variável testada

✓ Algoritmo

Algoritmo Seleção Múltipla

Leia opção

Caso opção = 1, então

Imprima “opção 1 foi acionada”

Fim Opção

Caso opção = 2, então

Imprima “opção 2 foi acionada”

Fim Opção

Caso opção = 3, então

Imprima “opção 3 foi acionada”

Fim Opção

Fim Algoritmo

Estruturas de Controle em JAVA

- Estrutura de seleção múltipla switch

```
import javax.swing.JOptionPane;
public class SelecaoMultipla {
    public static void main (String args[] ) {
        int opcao=9999; // Define e inicializa a Opção a ser entrada pelo usuário
        // Processamento
        while ( opcao != 0 ) { // Enquanto opção não for igual a 0, continua execução
            String resposta = JOptionPane.showInputDialog("Qual a sua opção (0 para sair; 1-3 para teste) ? ");
            // converte a resposta para inteiro a fim de poder ser atribuído à opcao
            opcao = Integer.parseInt(resposta);
            switch (opcao) {
                case 1:
                    JOptionPane.showMessageDialog (null, "A Opção 1 foi acionada! ");
                    break;
                case 2:
                    JOptionPane.showMessageDialog (null, "A Opção 2 foi acionada! ");
                    break;
                case 3:
                    JOptionPane.showMessageDialog (null, "A Opção 3 foi acionada! ");
                    break;
                default:
                    JOptionPane.showMessageDialog (null, "Informe 0 para sair ou valores de 1 a 3 para teste! ");
                    break;
            } // Fim do Switch
        } // Fim do While
        System.exit( 0 );
    }
}
```

Estruturas de Controle em JAVA

- Estrutura de seleção múltipla switch

```
import javax.swing.JOptionPane;
```

```
public class SelecaoMultipla {
```

```
    public static void main (String args[] ) {
```

```
        int opcao=9999; // Define e inicializa a Opção a ser entrada pelo usuário
```

```
        // Processamento
```

```
        while ( opcao != 0 ) { // Enquanto opção não for igual a 0, continua execução
```

```
            String resposta = JOptionPane.showInputDialog("Qual a sua opção (0 para sair; 1-3 para teste) ? ");
```

```
            // converte a resposta para inteiro a fim de poder ser atribuído à opcao
```

```
            opcao = Integer.parseInt(resposta);
```

```
            switch (opcao) {
```

```
                case 1:
```

```
                    JOptionPane.showMessageDialog (null, "A Opção 1 foi acionada! ");
```

```
                    break;
```

```
                case 2:
```

```
                    JOptionPane.showMessageDialog (null, "A Opção 2 foi acionada! ");
```

```
                    break;
```

```
                case 3:
```

```
                    JOptionPane.showMessageDialog (null, "A Opção 3 foi acionada! ");
```

```
                    break;
```

```
                default:
```

```
                    JOptionPane.showMessageDialog (null, "Informe 0 para sair ou valores de 1 a 3 para teste! ");
```

```
                    break;
```

```
            } // Fim do Switch
```


```
        } // Fim do While
```

```
        System.exit( 0 );
```

```
    }
```

```
}
```

Executa enquanto opção
não for igual a 0



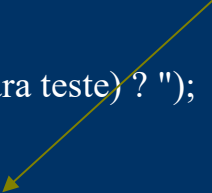
Estruturas de Controle em JAVA

- Estrutura de seleção múltipla switch

```
import javax.swing.JOptionPane;
public class SelecaoMultipla {
    public static void main (String args[] ) {
        int opcao=9999; // Define e inicializa a Opção a ser entrada pelo usuário
        // Processamento
        while ( opcao != 0 ) { // Enquanto opção não for igual a 0, continua execução
            String resposta = JOptionPane.showInputDialog("Qual a sua opção (0 para sair; 1-3 para teste) ? ");
            // converte a resposta para inteiro a fim de poder ser atribuído à opcao
            opcao = Integer.parseInt(resposta);

            switch (opcao) {
                case 1:
                    JOptionPane.showMessageDialog (null, "A Opção 1 foi acionada! ");
                    break;
                case 2:
                    JOptionPane.showMessageDialog (null, "A Opção 2 foi acionada! ");
                    break;
                case 3:
                    JOptionPane.showMessageDialog (null, "A Opção 3 foi acionada! ");
                    break;
                default:
                    JOptionPane.showMessageDialog (null, "Informe 0 para sair ou valores de 1 a 3 para teste! ");
                    break;
            } // Fim do Switch
        } // Fim do While
        System.exit( 0 );
    }
}
```

De acordo com o valor informado à variável **opcao** executa blocos distintos de código

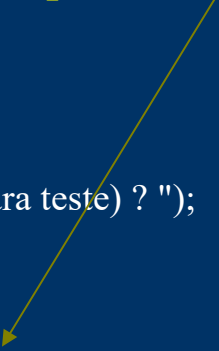


Estruturas de Controle em JAVA

- Estrutura de seleção múltipla switch

```
import javax.swing.JOptionPane;
public class SelecaoMultipla {
    public static void main (String args[] ) {
        int opcao=9999; // Define e inicializa a Opção a ser entrada pelo usuário
        // Processamento
        while ( opcao != 0 ) { // Enquanto opção não for igual a 0, continua execução
            String resposta = JOptionPane.showInputDialog("Qual a sua opção (0 para sair; 1-3 para teste) ? ");
            // converte a resposta para inteiro a fim de poder ser atribuído à opcao
            opcao = Integer.parseInt(resposta);
            switch (opcao) {
                case 1:
                    JOptionPane.showMessageDialog (null, "A Opção 1 foi acionada! ");
                    break;
                case 2:
                    JOptionPane.showMessageDialog (null, "A Opção 2 foi acionada! ");
                    break;
                case 3:
                    JOptionPane.showMessageDialog (null, "A Opção 3 foi acionada! ");
                    break;
                default:
                    JOptionPane.showMessageDialog (null, "Informe 0 para sair ou valores de 1 a 3 para teste! ");
                    break;
            } // Fim do Switch
        } // Fim do While
        System.exit( 0 );
    }
}
```

Executa se a variável
opcao for igual a 1

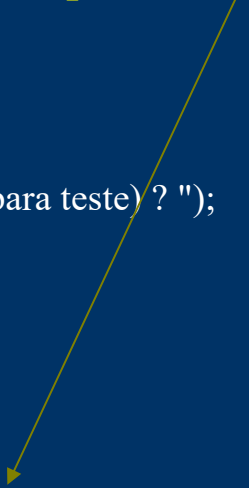


Estruturas de Controle em JAVA

- Estrutura de seleção múltipla switch

```
import javax.swing.JOptionPane;
public class SelecaoMultipla {
    public static void main (String args[] ) {
        int opcao=9999; // Define e inicializa a Opção a ser entrada pelo usuário
        // Processamento
        while ( opcao != 0 ) { // Enquanto opção não for igual a 0, continua execução
            String resposta = JOptionPane.showInputDialog("Qual a sua opção (0 para sair; 1-3 para teste) ? ");
            // converte a resposta para inteiro a fim de poder ser atribuído à opcao
            opcao = Integer.parseInt(resposta);
            switch (opcao) {
                case 1:
                    JOptionPane.showMessageDialog (null, "A Opção 1 foi acionada! ");
                    break;
                case 2:
                    JOptionPane.showMessageDialog (null, "A Opção 2 foi acionada! ");
                    break;
                case 3:
                    JOptionPane.showMessageDialog (null, "A Opção 3 foi acionada! ");
                    break;
                default:
                    JOptionPane.showMessageDialog (null, "Informe 0 para sair ou valores de 1 a 3 para teste! ");
                    break;
            } // Fim do Switch
        } // Fim do While
        System.exit( 0 );
    }
}
```

Executa se a variável
opcao for igual a 2

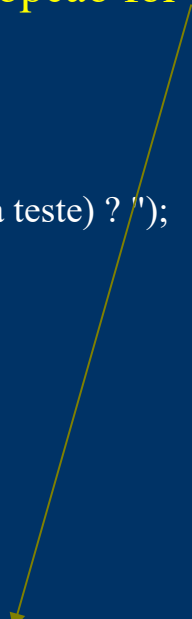


Estruturas de Controle em JAVA

- Estrutura de seleção múltipla switch

```
import javax.swing.JOptionPane;
public class SelecaoMultipla {
    public static void main (String args[] ) {
        int opcao=9999; // Define e inicializa a Opção a ser entrada pelo usuário
        // Processamento
        while ( opcao != 0 ) { // Enquanto opção não for igual a 0, continua execução
            String resposta = JOptionPane.showInputDialog("Qual a sua opção (0 para sair; 1-3 para teste) ? ");
            // converte a resposta para inteiro a fim de poder ser atribuído à opcao
            opcao = Integer.parseInt(resposta);
            switch (opcao) {
                case 1:
                    JOptionPane.showMessageDialog (null, "A Opção 1 foi acionada! ");
                    break;
                case 2:
                    JOptionPane.showMessageDialog (null, "A Opção 2 foi acionada! ");
                    break;
                case 3:
                    JOptionPane.showMessageDialog (null, "A Opção 3 foi acionada! ");
                    break;
                default:
                    JOptionPane.showMessageDialog (null, "Informe 0 para sair ou valores de 1 a 3 para teste! ");
                    break;
            } // Fim do Switch
        } // Fim do While
        System.exit( 0 );
    }
}
```

Executa se a variável
opcao for igual a 3




Estruturas de Controle em JAVA

- Estrutura de seleção múltipla switch

```
import javax.swing.JOptionPane;
public class SelecaoMultipla {
    public static void main (String args[] ) {
        int opcao=9999; // Define e inicializa a Opção a ser entrada pelo usuário
        // Processamento
        while ( opcao != 0 ) { // Enquanto opção não for igual a 0, continua execução
            String resposta = JOptionPane.showInputDialog("Qual a sua opção (0 para sair; 1-3 para teste) ? ");
            // converte a resposta para inteiro a fim de poder ser atribuído à opcao
            opcao = Integer.parseInt(resposta);
            switch (opcao) {
                case 1:
                    JOptionPane.showMessageDialog (null, "A Opção 1 foi acionada! ");
                    break;
                case 2:
                    JOptionPane.showMessageDialog (null, "A Opção 2 foi acionada! ");
                    break;
                case 3:
                    JOptionPane.showMessageDialog (null, "A Opção 3 foi acionada! ");
                    break;
                default:
                    JOptionPane.showMessageDialog (null, "Informe 0 para sair ou valores de 1 a 3 para teste! ");
                    break;
            } // Fim do Switch
        } // Fim do While
        System.exit( 0 );
    }
}
```

Executa se qualquer outro valor for atribuído à variável opcao



Estruturas de Controle em JAVA

- Estrutura de repetição do/while
 - ✓ É semelhante à estrutura while
 - Na estrutura while a condição de continuação do laço é testada no começo, antes do corpo do laço ser executado
 - ✓ A estrutura do/while testa a condição de continuação do laço no final, depois do corpo do laço ter sido executado
 - O corpo do laço é executado pelo menos uma vez
 - ✓ Algoritmo

Faça

instrução;

Enquanto (condição for verdadeira);

✓ JAVA

do

// instrução

while (x < 10);

condição



✓ JAVA

do {

// instrução 1

// instrução 2 ...

} while (x < 10);

Bloco de
instruções
(comandos)



Estruturas de Controle em JAVA

- Estrutura de repetição do/while

✓ Exercício

→ Faça um algoritmo, e a respectiva implementação em JAVA, para imprimir na tela os números de 1000 a 1

✓ Resposta

Algoritmo

 inicialize contador = 1000;

 faça

 Imprima contador;

 contador = contador – 1;

 enquanto (contador \leq 0);

Fim Algoritmo

Estruturas de Controle em JAVA

- Estrutura de repetição do/while

✓ Implementação em JAVA

```
public class Repeticao {  
    public static void main (String args[]) {  
        int contador = 1000; // Declara e inicializa a variável contador  
        do {  
            System.out.println(contador);  
            contador = contador - 1;  
  
        } while (contador != 0);  
    }  
}
```


Estruturas de Controle em JAVA

- Instruções break e continue
 - ✓ Alteram o fluxo de controle
 - ✓ A instrução break quando executada em uma estrutura while, for, do/while ou switch ocasiona a saída imediata da estrutura
 - A execução continua com a primeira instrução depois da estrutura
 - ✓ Algoritmo

Algoritmo Teste da Instrução Break

Para contador = 1, execute o bloco de comandos e incremente contador em 1 até que seja igual a 10

Se contador igual a 5, então

Pare

Fim Se

Imprima contador

Fim Para

Fim Algoritmo

Estruturas de Controle em JAVA

- Instruções break e continue
- ✓ Implementação em JAVA do Algoritmo para teste instrução Break

```
public class Pare {  
    public static void main (String args[]) {  
        int contador; // Declara a variável contador  
        for (contador = 1; contador <= 10; contador = contador + 1) {  
            if (contador == 5) {  
                break; // Quando contador for 5, o restante da execução  
                       // do laço for é desconsiderado  
                       // Sem a instrução Break o laço executaria até  
                       // contador igual a 10  
            }  
            System.out.println(contador);  
        }  
    }  
}
```

Estruturas de Controle em JAVA

- Instruções break e continue
 - ✓ A instrução continue quando executada em uma estrutura while, for, do/while, pula as instruções restantes no corpo e prossegue com a próxima iteração do laço
 - ✓ Nas estruturas while e do/while, o teste de continuação do laço é avaliado imediatamente depois da instrução continue ter sido executada

Algoritmo Teste da Instrução Continue

Para contador = 1, execute o bloco de comandos e incremente contador em 1 até que seja igual a 10

Se contador igual a 5, então

Pule

Fim Se

Imprima contador

Fim Para

Fim Algoritmo

Estruturas de Controle em JAVA

- Instruções break e continue
- ✓ Implementação em JAVA do Algoritmo para teste da instrução continue

```
public class Pule {  
    public static void main (String args[]) {  
        int contador; // Declara a variável contador  
        for (contador = 1; contador <= 10; contador = contador + 1) {  
            if (contador == 5) {  
                continue; // Pula a execução do laço para a próxima  
                           // iteração sem executar a instrução  
                           // System.out.println(contador)  
                           // A saída não imprime 5  
            }  
            System.out.println(contador);  
        }  
    }  
}
```

Estruturas de Controle em JAVA

- Instruções break e continue
- ✓ Implementação em JAVA do Algoritmo para teste da instrução continue

```
public class Pule {  
    public static void main (String args[]) {  
        int contador; // Declara a variável contador  
        for (contador = 1; contador <= 10; contador = contador + 1) {  
            if (contador == 5) {  
                continue; // Pula a execução do laço para a próxima  
                           // iteração sem executar a instrução  
                           // System.out.println(contador)  
                           // A saída não imprime 5  
            }  
            System.out.println(contador);  
        }  
    }  
}
```

Operadores em JAVA

- Atribuição
- Incremento e Decremento
- Lógicos



Operadores em JAVA

- Atribuição

- ✓ Java fornece vários operadores de atribuição para abreviar expressões de atribuição

- `c = c + 3;`

Pode ser representado por

- `c += 3;`

- ✓ O operador `+=` adiciona o valor da expressão à direita do operador ao valor da variável à esquerda do operador, armazenando o resultado na variável à esquerda do operador. Assim:

- Variável = variável operador expressão;

Pode ser abreviado para

- Variável operador = expressão;

- ✓ O operador é um dos operadores binários

- `+`

- `-`

soma

- `/`

subtração

- `*`

divisão

- `/`

multiplicação

- `%`

divisão

módulo ou resto da divisão

Operadores em JAVA

- Atribuição
- ✓ Pressuponha: `int c=3, d=5, e=4, f=6, g=12;`

Operador de atribuição	Expressão	Explicação	Resultado
<code>+=</code>	<code>c += 7</code>		
<code>-=</code>	<code>d -= 4</code>		
<code>=</code>	<code>e *= 5</code>		
<code>/=</code>	<code>f /= 3</code>		
<code>%=</code>	<code>g %= 9</code>		

Operadores em JAVA

- Atribuição

✓ Pressuponha: `int c=3, d=5, e=4, f=6, g=12;`

Operador de atribuição	Expressão	Explicação	Resultado
<code>+=</code>	<code>c += 7</code>	<code>c = c + 7</code>	10
<code>-=</code>	<code>d -= 4</code>	<code>d = d - 4</code>	1
<code>=</code>	<code>e *= 5</code>	<code>e = e * 5</code>	20
<code>/=</code>	<code>f /= 3</code>	<code>f = f / 3</code>	2
<code>%=</code>	<code>g %= 9</code>	<code>g = g % 9</code>	3

Operadores em JAVA

- Incremento e Decremento
 - ✓ Java fornece o operador de incremento unário ++ e o operador de decremento unário --
 - ✓ Pré-incremento
 - Caso o operador de incremento seja colocado antes de uma variável
 - ✓ Pré-decremento
 - Caso o operador de decremento seja colocado antes de uma variável
 - ✓ Pós-incremento
 - Caso o operador de incremento seja colocado depois de uma variável
 - ✓ Pós-decremento
 - Caso o operador de decremento seja colocado depois de uma variável
-
-

Operadores em JAVA

- Incremento e Decremento

Operador	Denominação	Expressão	Explicação
++	pré-incremento	++a	incrementa a em 1 e usa o seu valor
++	pós-incremento	a++	usa o valor de a e o incrementa em 1
--	pré-decremento	--b	decrementa b em 1 e usa o seu valor
--	pós-decremento	b--	usa o valor de b e o decrementa em 1

Operadores em JAVA

- Incremento e Decremento

```
public class Incremento {  
    public static void main (String args[]) {  
        int c;  
        c = 5;  
        System.out.println( c ); // imprime 5  
        System.out.println( c++ ); // imprime 5 então pós-incrementa  
        System.out.println( c ); // imprime 6  
  
        System.out.println();  
  
        c = 5;  
        System.out.println( c ); // imprime 5  
        System.out.println( ++c ); // pré-incrementa e imprime 6  
        System.out.println( c ); // imprime 6  
  
    }  
  
}
```

Operadores em JAVA

- Lógicos

- ✓ Condições simples

- ➔ Relacionais

> Maior que

< Menor que

>= Maior que ou igual a

<= Menor que ou igual a

- ➔ De Igualdade

== Igual a

!= Não igual a (diferente)

- ✓ Condições complexas

&& AND lógico

& AND lógico booleano

|| OR lógico

| OR lógico booleano inclusivo


^ OR lógico booleano exclusivo

! NOT lógico também conhecido como negação lógica

Operadores em JAVA

- Lógicos
- ✓ Observe a amostra de código:

Condição complexa



```
if (sexo.equals("F") && idade >= 65) {  
    contador++;  
} else {  
    outros++;  
}
```

Operadores em JAVA

- Lógicos
- ✓ Observe a amostra de código:

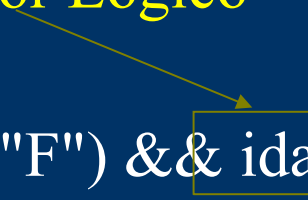
Expressão 1 Expressão 2

```
if ( sexo.equals("F") && idade >= 65 ) {  
    contador++;  
} else {  
    outros++;  
}
```

Operadores em JAVA

- Lógicos
- ✓ Observe a amostra de código:

Operador Lógico



```
if ( sexo.equals("F") && idade >= 65 ) {  
    contador++;  
} else {  
    outros++;  
}
```


Operadores em JAVA

- Lógicos
- ✓ Observe a amostra de código:

Expressão 1

Expressão 2

False

True

```
if ( sexo.equals("F") && idade >= 65 ) {  
    contador++;  
} else {  
    outros++;  
}
```

Operadores em JAVA

- Lógicos
- ✓ Observe a amostra de código:

Resultado

False

```
if ( sexo.equals("F") && idade >= 65 ) {  
    contador++;  
} else {  
    outros++;  
}
```

Executa



Operadores em JAVA

- Lógicos
- ✓ Observe a amostra de código:

Expressão 1

Expressão 2

True

True

```
if ( sexo.equals("F") && idade >= 65 ) {  
    contador++;  
} else {  
    outros++;  
}
```

Operadores em JAVA

- Lógicos
- ✓ Observe a amostra de código:

Resultado

True

```
if ( sexo.equals("F") && idade >= 65 ) {  
    contador++;  
} else {  
    outros++;  
}
```

Executa

Operadores em JAVA

- Lógicos

✓ Expressão 1	Expressão 2	Expressão 1 && Expressão 2
false	false	false
false	true	false
true	false	false
true	true	true

✓ Expressão 1	Expressão 2	Expressão 1 Expressão 2
false	false	false
false	true	true
true	false	true
true	true	true

✓ Expressão 1	Expressão 2	Expressão 1 ^ Expressão 2
false	false	false
false	true	true
true	false	true
true	true	false

Operadores em JAVA

- Lógicos
- ✓ Algoritmo

Programa para calcular a quantidade de mulheres da terceira idade

Leia contador, outros;

Faça

 Leia sexo;

 Leia idade;

 Se (sexo = F e idade \geq 65) então

 contador = contador + 1;

 Senão

 outros = outros + 1;

 Fim Senão

 Fim Se

Enquanto(idade \neq 0)

 Imprima contador;

 Imprima outros;

Fim Programa;

Condição Complexa



Operadores em JAVA

- Implementação

```
public class Logico1 {  
    public static void main (String args[]) {  
        String sexo;  
        int idade;  
        int contador=0,outros=0;  
        System.out.println("Programa para contar a quantidade de mulheres da terceira idade!");  
        System.out.println("Para sair informe 0 para a idade");  
        System.out.println();  
  
        do {  
            System.out.println("Informe o sexo (M para Masculino) ou (F para Feminino) ");  
            sexo = Util.readStr().trim();  
            System.out.println("Informe a idade ");  
            idade = Util.readInt();  
  
            if (sexo.equals("F") && idade >= 65) {  
                contador++;  
            } else {  
                outros++;  
            }  
        } while (idade != 0);  
        System.out.println();  
        System.out.println("A quantidade de mulheres da terceira idade é: " + contador);  
        System.out.println("Não são mulheres da terceira idade: " + outros);  
    }  
}
```

Operadores em JAVA

• Implementação

```
public class Logico1 {  
    public static void main (String args[]) {  
        String sexo;  
        int idade;  
        int contador=0,outros=0;  
        System.out.println("Programa para contar a quantidade de mulheres da terceira idade!");  
        System.out.println("Para sair informe 0 para a idade");  
        System.out.println();  
  
        do {  
            System.out.println("Informe o sexo (M para Masculino) ou (F para Feminino) ");  
            sexo = Util.readStr().trim();  
            System.out.println("Informe a idade ");  
            idade = Util.readInt();  
  
            if (sexo.equals("F") && idade >= 65) {  
                contador++;  
            } else {  
                outros++;  
            }  
        } while (idade != 0);  
        System.out.println();  
        System.out.println("A quantidade de mulheres da terceira idade é: " + contador);  
        System.out.println("Não são mulheres da terceira idade: " + outros);  
    }  
}
```

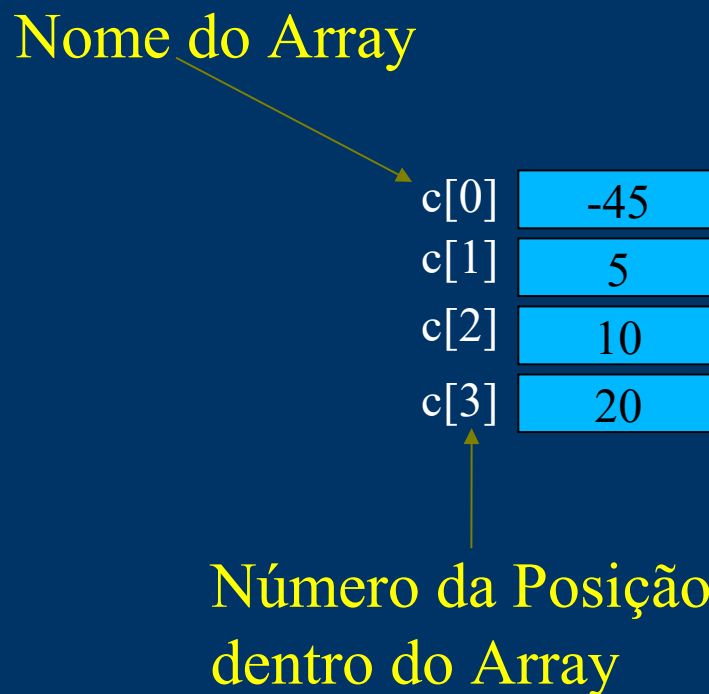
Condição Complexa

Comparação da String "F" com o valor da variável String sexo

Arrays

- É um grupo de posições contíguas na memória que possuem o mesmo nome e o mesmo tipo
- Para nos referirmos a uma particular elemento do array, especificamos o nome do array e o número da posição do elemento particular no array.

Nome do Array



The diagram illustrates an array structure. It consists of a vertical list of four elements, each with an index label on the left and a value in a blue box on the right. The labels are c[0], c[1], c[2], and c[3]. The values are -45, 5, 10, and 20. A yellow arrow points from the text 'Nome do Array' to the 'c' part of the c[0] label. Another yellow arrow points from the text 'Número da Posição dentro do Array' to the '3' part of the c[3] label.

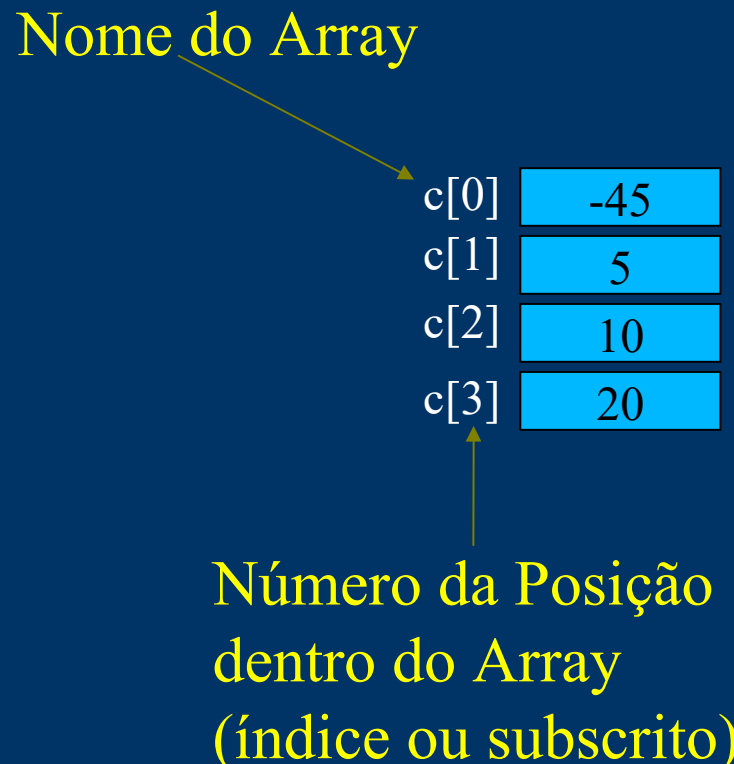
c[0]	-45
c[1]	5
c[2]	10
c[3]	20

Número da Posição
dentro do Array

Arrays

- Os elementos do array podem ser referenciado fornecendo-se o nome do array seguido pelo número da posição do elemento particular entre colchetes “[” e “]”
- O primeiro elemento de um array localiza-se na posição 0, enquanto que o último elemento localiza-se na posição (tamanho do array - 1)

Nome do Array



c[0]	-45
c[1]	5
c[2]	10
c[3]	20

Número da Posição
dentro do Array
(índice ou subscrito)

Arrays

- Manipulação de Arrays


- ✓ Dado que $\text{int } a = 1$ e $\text{int } b = 2$

- ✓ A expressão $c[a + b] += 2$ teria como resultado a atribuição de 22 ao elemento $c[3]$ do array dado no exemplo

- $c[3] = c[3] + 2$

- $c[3] = 20 + 2 \iff c[3] = 22$

Nome do Array



c[0]	-45
c[1]	5
c[2]	10
c[3]	20

Número da Posição
dentro do Array
(índice ou subscrito)

Arrays

- Manipulação de Arrays
 - ✓ O comprimento do array é determinado pela expressão `c.length`
 - No exemplo dado `c.length` retorna 4
 - ✓ Quaisquer operações podem ser realizadas sobre os elementos de um array como se faz com variáveis simples
 - `int soma = c[0] + c[1] + c[2] + c[3];`
 - Cujo resultado seria a atribuição de -10 à variável inteira soma

Nome do Array

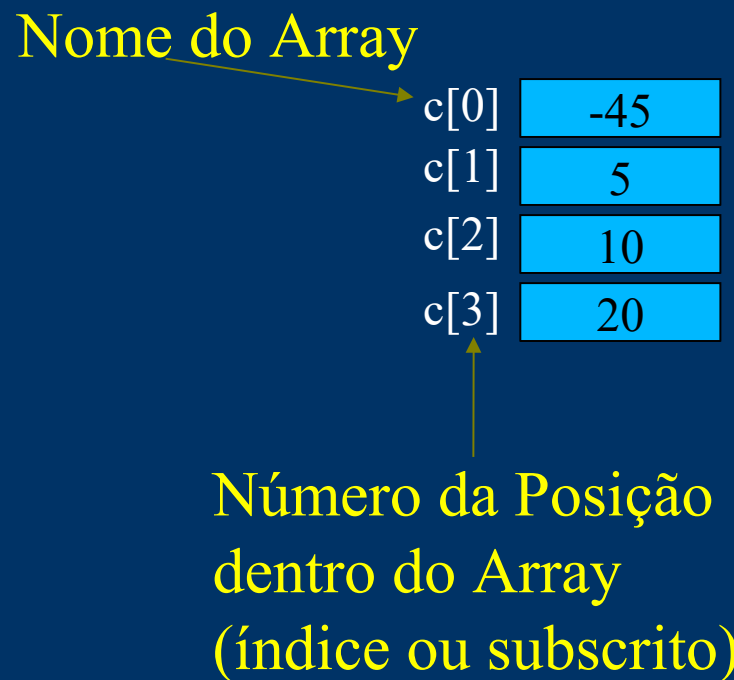
c[0]	-45
c[1]	5
c[2]	10
c[3]	20

Número da Posição
dentro do Array
(índice ou subscrito)

Arrays

- Declaração e alocação de arrays
 - ✓ Precisa-se especificar o tipo dos elementos
 - ✓ Usa-se o operador new uma vez que os arrays são objetos
- `int c[] = new int [4];`

Nome do Array



c[0]	-45
c[1]	5
c[2]	10
c[3]	20

Número da Posição
dentro do Array
(índice ou subscrito)

Arrays

- Declaração e alocação de arrays
 - ✓ `int c[];` // Declara o array
 - ✓ `c = new int [4];` // Aloca 4 elementos do tipo inteiro para o array

Nome do Array

c[0]	-45
c[1]	5
c[2]	10
c[3]	20

Número da Posição
dentro do Array
(índice ou subscrito)

Arrays

- Exemplo de implementação

```
public class Array {  
    public static void main (String args[]) {  
        int c[] = new int[4]; // Declaração e alocação do array  
        c[0] = -45;  
        c[1] = 5;  
        c[2] = 10;  
        c[3] = 20;  
        int soma = c[0] + c[1] + c[2] + c[3];  
  
        System.out.println("Os elementos do array são: ");  
        System.out.println("c[0] ..... " + c[0]);  
        System.out.println("c[1] ..... " + c[1]);  
        System.out.println("c[2] ..... " + c[2]);  
        System.out.println("c[3] ..... " + c[3]);  
        System.out.println("");  
        System.out.println("A Soma é: " + soma);  
        System.out.println("");  
  
        int a = 1, b=2;  
        c[a+b] += 2;  
  
        System.out.println("O resultado da expressão c[a + b] += 2, dado que a é 1 e b é 2, torna c[3] = " + c[3]);  
    }  
}
```

Arrays

- Exemplo de implementação

```
public class Array {  
    public static void main (String args[]) {  
        int c[] = new int[4]; // Declaração e alocação do array  
        c[0] = -45;  
        c[1] = 5;  
        c[2] = 10;  
        c[3] = 20;  
        int soma = c[0] + c[1] + c[2] + c[3];  
  
        System.out.println("Os elementos do array são: ");  
        System.out.println("c[0] ..... " + c[0]);  
        System.out.println("c[1] ..... " + c[1]);  
        System.out.println("c[2] ..... " + c[2]);  
        System.out.println("c[3] ..... " + c[3]);  
        System.out.println("");  
        System.out.println("A Soma é: " + soma);  
        System.out.println("");  
  
        int a = 1, b=2;  
        c[a+b] += 2;  
  
        System.out.println("O resultado da expressão c[a + b] += 2, dado que a é 1 e b é 2, torna c[3] = " + c[3]);  
    }  
}
```

Declaração e alocação do array

Arrays

- Exemplo de implementação

```
public class Array {  
    public static void main (String args[]) {  
        int c[] = new int[4]; // Declaração e alocação do array  
        c[0] = -45;  
        c[1] = 5;  
        c[2] = 10;  
        c[3] = 20;  
        int soma = c[0] + c[1] + c[2] + c[3];  
  
        System.out.println("Os elementos do array são: ");  
        System.out.println("c[0] ..... " + c[0]);  
        System.out.println("c[1] ..... " + c[1]);  
        System.out.println("c[2] ..... " + c[2]);  
        System.out.println("c[3] ..... " + c[3]);  
        System.out.println("");  
        System.out.println("A Soma é: " + soma);  
        System.out.println("");  
  
        int a = 1, b=2;  
        c[a+b] += 2;  
  
        System.out.println("O resultado da expressão c[a + b] += 2, dado que a é 1 e b é 2, torna c[3] = " + c[3]);  
    }  
}
```

Atribuição de valores inteiros
aos elementos do array

Arrays

- Exemplo de implementação

```
public class Array {  
    public static void main (String args[]) {  
        int c[] = new int[4]; // Declaração e alocação do array  
        c[0] = -45;  
        c[1] = 5;  
        c[2] = 10;  
        c[3] = 20;  
        int soma = c[0] + c[1] + c[2] + c[3];  
  
        System.out.println("Os elementos do array são: ");  
        System.out.println("c[0] ..... " + c[0]);  
        System.out.println("c[1] ..... " + c[1]);  
        System.out.println("c[2] ..... " + c[2]);  
        System.out.println("c[3] ..... " + c[3]);  
        System.out.println("");  
        System.out.println("A Soma é: " + soma);  
        System.out.println("");  
  
        int a = 1, b=2;  
        c[a+b] += 2;  
  
        System.out.println("O resultado da expressão c[a + b] += 2, dado que a é 1 e b é 2, torna c[3] = " + c[3]);  
    }  
}
```

Criação da variável inteira soma para receber a soma dos elementos do array

Arrays

- Exemplo de implementação

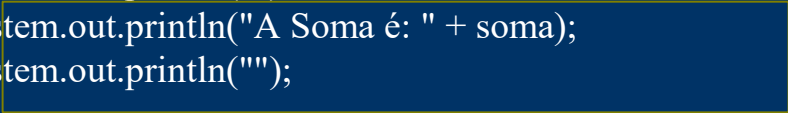
```
public class Array {  
    public static void main (String args[]) {  
        int c[] = new int[4]; // Declaração e alocação do array  
        c[0] = -45;  
        c[1] = 5;  
        c[2] = 10;  
        c[3] = 20;  
        int soma = c[0] + c[1] + c[2] + c[3];  
  
        System.out.println("Os elementos do array são: ");  
        System.out.println("c[0] ..... " + c[0]);  
        System.out.println("c[1] ..... " + c[1]);  
        System.out.println("c[2] ..... " + c[2]);  
        System.out.println("c[3] ..... " + c[3]);  
        System.out.println("");  
        System.out.println("A Soma é: " + soma);  
        System.out.println("");  
  
        int a = 1, b=2;  
        c[a+b] += 2;  
  
        System.out.println("O resultado da expressão c[a + b] += 2, dado que a é 1 e b é 2, torna c[3] = " + c[3]);  
    }  
}
```

Impresão dos elementos do array

Arrays

- Exemplo de implementação

```
public class Array {  
    public static void main (String args[]) {  
        int c[] = new int[4]; // Declaração e alocação do array  
        c[0] = -45;  
        c[1] = 5;  
        c[2] = 10;  
        c[3] = 20;  
        int soma = c[0] + c[1] + c[2] + c[3];  
  
        System.out.println("Os elementos do array são: ");  
        System.out.println("c[0] ..... " + c[0]);  
        System.out.println("c[1] ..... " + c[1]);  
        System.out.println("c[2] ..... " + c[2]);  
        System.out.println("c[3] ..... " + c[3]);  
        System.out.println("");  
        System.out.println("A Soma é: " + soma);  
        System.out.println("");  
  
        int a = 1, b=2;  
        c[a+b] += 2;  
  
        System.out.println("O resultado da expressão c[a + b] += 2, dado que a é 1 e b é 2, torna c[3] = " + c[3]);  
    }  
}
```



Impressão do valor atribuído à
variável soma

Arrays

- Exemplo de implementação

```
public class Array {  
    public static void main (String args[]) {  
        int c[] = new int[4]; // Declaração e alocação do array  
        c[0] = -45;  
        c[1] = 5;  
        c[2] = 10;  
        c[3] = 20;  
        int soma = c[0] + c[1] + c[2] + c[3];
```

```
        System.out.println("Os elementos do array são: ");  
        System.out.println("c[0] ..... " + c[0]);  
        System.out.println("c[1] ..... " + c[1]);  
        System.out.println("c[2] ..... " + c[2]);  
        System.out.println("c[3] ..... " + c[3]);  
        System.out.println("");  
        System.out.println("A Soma é: " + soma);  
        System.out.println("");
```

```
        int a = 1, b=2;  
        c[a+b] += 2;
```

```
        System.out.println("O resultado da expressão c[a + b] += 2, dado que a é 1 e b é 2, torna c[3] = " + c[3]);
```

```
    }  
}
```

Uso do operador de atribuição
+= a um elemento do array



Arrays

- Exemplo de implementação

```
public class Array {  
    public static void main (String args[]) {  
        int c[] = new int[4]; // Declaração e alocação do array  
        c[0] = -45;  
        c[1] = 5;  
        c[2] = 10;  
        c[3] = 20;  
        int soma = c[0] + c[1] + c[2] + c[3];  
  
        System.out.println("Os elementos do array são: ");  
        System.out.println("c[0] ..... " + c[0]);  
        System.out.println("c[1] ..... " + c[1]);  
        System.out.println("c[2] ..... " + c[2]);  
        System.out.println("c[3] ..... " + c[3]);  
        System.out.println("");  
        System.out.println("A Soma é: " + soma);  
        System.out.println("");  
  
        int a = 1, b=2;  
        c[a+b] += 2;  
  
        System.out.println("O resultado da expressão c[a + b] += 2, dado que a é 1 e b é 2, torna c[3] = " + c[3]);  
    }  
}
```

Impresão do novo valor alocado
ao elemento c[3] do array c
após a execução do operador de
atribuição +=



System.out.println("O resultado da expressão c[a + b] += 2, dado que a é 1 e b é 2, torna c[3] = " + c[3]);

Arrays

- Declarando e Alocando Arrays
 - ✓ Para se declarar vários arrays de um mesmo tipo de elementos, usa-se
 - `double[] array1, array2;`
 - ✓ Para se declarar e alocar vários arrays de um mesmo tipo de elementos:
 - `double[] array1 = new double[10], array2 = new double[20];`
 - ✓ Os elementos de um array podem ser alocados e inicializados na declaração de array seguindo a declaração com um sinal de igual e uma lista inicializadora separada por vírgulas entre chaves “{“ e “}”
 - `int n[] = { 10,20, 30, 40, 50 };`
 - ♦ Cria um array de 5 elementos com os índices 0, 1, 2, 3, 4
 - ♦ Não precisa do operador new que é fornecido automaticamente pelo compilador

Arrays

- Passando arrays para métodos

- ✓ Declaração e alocação

```
int temperaturaPorHora[] = new int[24];
```

- ✓ Chamada de Método

```
modificaArray( temperaturaPorHora );
```

- ✓ Declaração de Método

```
void modificaArray( int b[] )
```

- ✓ Chamada de Método com retorno de um array de inteiros

```
int[] modificaArray( temperaturaPorHora );
```

- ✓ Declaração de Método

```
int[] modificaArray( int b[] )
```

Arrays

- Arrays Multimensionais
 - ✓ Arrays com dois índices são utilizados freqüentemente para representar tabelas de valores organizadas em linhas e colunas
 - ✓ Para se identificar um elemento da tabela específico, devemos especificar os dois índices (subscritos)
 - ➔ Por convenção o primeiro identifica a linha do elemento e o segundo a coluna (arrays bidimensionais)

	Coluna 0	Coluna 1	Coluna 2	Coluna 3
Linha 0	b[0][0]	b[0][1]	b[0][2]	b[0][3]
Linha 1	b[1][0]	b[1][1]	b[1][2]	b[1][3]
Linha 2	b[2][0]	b[2][1]	b[2][2]	b[2][3]

Índice da coluna

Índice da linha

Nome do Array

Arrays

- Arrays Multimensionais

✓ Definição e inicialização de Arrays multidimensionais

```
int b[] [];
```

```
b = new int[ 3 ] [ 4 ];
```

	Coluna 0	Coluna 1	Coluna 2	Coluna 3
Linha 0	b[0][0]	b[0][1]	b[0][2]	b[0][3]
Linha 1	b[1][0]	b[1][1]	b[1][2]	b[1][3]
Linha 2	b[2][0]	b[2][1]	b[2][2]	b[2][3]

Índice da coluna

Índice da linha

Nome do Array

Arrays

- Arrays Multidimensionais

- ✓ Um array bidimensional `b [2] [2]` poderia ser declarado e inicializado com
`int b[] [] = { { 1, 2 }, { 3, 4 } }`

- ✓ O array de inteiros `b` com a linha 0 contendo os elementos (1 e 2) e a linha 1 contendo os elementos (3,4 e 5) poderia ser inicializado
`int b[][] = { { 1, 2 }, { 3, 4, 5 } };`

Arrays

- Arrays Multidimensionais

✓ Linhas e colunas de um array bidimensional podem ser alocadas em momentos distintos

```
int b[][];
```

```
b = new int [ 2 ][ ];
```

```
b [ 0 ] = new int[ 5 ];
```

```
b [ 1 ] = new int[ 3 ];
```

Aloca inicialmente duas linhas

Aloca 5 colunas para a linha 0

Aloca 3 colunas para a linha 1

Arrays

- Arrays Multidimensionais

```
public class ArrayMultidimensional {

    public void mostra(int a[][]) {
        for ( int i=0; i < a.length; i++ ) {
            for ( int j=0; j < a[i].length; j++ ) {
                System.out.print(a[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println();
    }

    public static void main(String args[]) {
        ArrayMultidimensional am = new ArrayMultidimensional();

        int array1[][] = { { 1,2,3 }, { 4,5,6 } };
        int array2[][] = { {1,2}, {3}, {4,5,6} };

        am.mostra(array1);
        am.mostra(array2);
    }
}
```

Arrays

- Arrays Multidimensionais

```
public class ArrayMultidimensional {
```

```
    public void mostra(int a[][]) {  
        for ( int i=0; i < a.length; i++ ) {  
            for ( int j=0; j < a[i].length; j++ ) {  
                System.out.print(a[i][j] + " ");  
            }  
            System.out.println();  
        }  
        System.out.println();  
    }  
}
```

← Método mostra recebe como parâmetro um array bidimensional e não retorna nada

```
    public static void main(String args[]) {  
        ArrayMultidimensional am = new ArrayMultidimensional();
```

```
        int array1[][] = { { 1,2,3 }, { 4,5,6 } };  
        int array2[][] = { { 1,2}, {3}, {4,5,6} };
```

```
        am.mostra(array1);  
        am.mostra(array2);
```

```
    }  
}
```

Arrays

- Arrays Multidimensionais

```
public class ArrayMultidimensional {
```

```
    public void mostra(int a[][]) {  
        for ( int i=0; i < a.length; i++ ) {  
            for ( int j=0; j < a[i].length; j++ ) {  
                System.out.print(a[i][j] + " ");  
            }  
            System.out.println();  
        }  
        System.out.println();  
    }  
}
```

Estrutura de repetição for para percorrer as linhas da matriz bidimensional

```
    public static void main(String args[]) {  
        ArrayMultidimensional am = new ArrayMultidimensional();
```

```
        int array1[][] = { { 1,2,3 }, { 4,5,6 } };  
        int array2[][] = { { 1,2}, {3}, {4,5,6} };
```

```
        am.mostra(array1);  
        am.mostra(array2);
```

```
    }  
}
```


Arrays

- Arrays Multidimensionais

```
public class ArrayMultidimensional {
```

```
    public void mostra(int a[][]) {  
        for ( int i=0; i < a.length; i++ ) {  
            for ( int j=0; j < a[i].length; j++ ) {  
                System.out.print(a[i][j] + " ");  
            }  
            System.out.println();  
        }  
        System.out.println();  
    }  
}
```

Estrutura de repetição for para percorrer as colunas da matriz bidimensional



```
    public static void main(String args[]) {  
        ArrayMultidimensional am = new ArrayMultidimensional();  
  
        int array1[][] = { { 1,2,3 }, { 4,5,6 } };  
        int array2[][] = { { 1,2}, {3}, {4,5,6} };  
  
        am.mostra(array1);  
        am.mostra(array2);  
    }  
}
```



Arrays

- Arrays Multidimensionais

```
public class ArrayMultidimensional {
```

```
    public void mostra(int a[][]) {  
        for ( int i=0; i < a.length; i++ ) {  
            for ( int j=0; j < a[i].length; j++ ) {  
                System.out.print(a[i][j] + " ");  
            }  
            System.out.println();  
        }  
        System.out.println();  
    }  
}
```

Dentro das estruturas repetitivas aninhadas, mostra o elemento da matriz correspondente à linha e coluna atuais



```
    public static void main(String args[]) {  
        ArrayMultidimensional am = new ArrayMultidimensional();  
  
        int array1[][] = { { 1,2,3 }, { 4,5,6 } };  
        int array2[][] = { { 1,2}, {3}, {4,5,6} };  
  
        am.mostra(array1);  
        am.mostra(array2);  
    }  
}
```

Arrays

- Arrays Multidimensionais

```
public class ArrayMultidimensional {
```

```
    public void mostra(int a[][]) {  
        for ( int i=0; i < a.length; i++ ) {  
            for ( int j=0; j < a[i].length; j++ ) {  
                System.out.print(a[i][j] + " ");  
            }  
            System.out.println();  
        }  
        System.out.println();  
    }  
}
```

Método principal main



```
    public static void main(String args[]) {  
        ArrayMultidimensional am = new ArrayMultidimensional();  
  
        int array1[][] = { { 1,2,3 }, { 4,5,6 } };  
        int array2[][] = { { 1,2 }, { 3 }, { 4,5,6 } };  
  
        am.mostra(array1);  
        am.mostra(array2);  
    }  
}
```

Arrays

- Arrays Multidimensionais

```
public class ArrayMultidimensional {
```

```
    public void mostra(int a[][]) {  
        for ( int i=0; i < a.length; i++ ) {  
            for ( int j=0; j < a[i].length; j++ ) {  
                System.out.print(a[i][j] + " ");  
            }  
            System.out.println();  
        }  
        System.out.println();  
    }  
}
```

Criação de um objeto am do tipo
ArrayMultidimensional



```
    public static void main(String args[]) {  
        ArrayMultidimensional am = new ArrayMultidimensional();
```

```
        int array1[][] = { { 1,2,3 }, { 4,5,6 } };  
        int array2[][] = { { 1,2}, {3}, {4,5,6} };
```

```
        am.mostra(array1);  
        am.mostra(array2);
```

```
    }  
}
```

Arrays

• Arrays Multidimensionais

```
public class ArrayMultidimensional {
```

```
    public void mostra(int a[][]) {  
        for ( int i=0; i < a.length; i++ ) {  
            for ( int j=0; j < a[i].length; j++ ) {  
                System.out.print(a[i][j] + " ");  
            }  
            System.out.println();  
        }  
        System.out.println();  
    }  
}
```


```
    public static void main(String args[]) {  
        ArrayMultidimensional am = new ArrayMultidimensional();
```

```
        int array1[][] = { { 1,2,3 }, { 4,5,6 } };  
        int array2[][] = { { 1,2}, {3}, {4,5,6} };
```

```
        am.mostra(array1);  
        am.mostra(array2);
```

```
    }  
}
```

Definição inicialização doas arrays array1 e array2




Arrays

- Arrays Multidimensionais

```
public class ArrayMultidimensional {  
  
    public void mostra(int a[][]) {  
        for ( int i=0; i < a.length; i++ ) {  
            for ( int j=0; j < a[i].length; j++ ) {  
                System.out.print(a[i][j] + " ");  
            }  
            System.out.println();  
        }  
        System.out.println();  
    }  
  
    public static void main(String args[]) {  
        ArrayMultidimensional am = new ArrayMultidimensional();  
  
        int array1[][] = { { 1,2,3 }, { 4,5,6 } };  
        int array2[][] = { { 1,2}, {3}, {4,5,6} };  
  
        am.mostra(array1);  
        am.mostra(array2);  
    }  
}
```

Acessa o método mostra disponível no objeto instanciado am, passando como prâmetro o array1




Arrays

- Arrays Multidimensionais

```
public class ArrayMultidimensional {  
  
    public void mostra(int a[][]) {  
        for ( int i=0; i < a.length; i++ ) {  
            for ( int j=0; j < a[i].length; j++ ) {  
                System.out.print(a[i][j] + " ");  
            }  
            System.out.println();  
        }  
        System.out.println();  
    }  
  
    public static void main(String args[]) {  
        ArrayMultidimensional am = new ArrayMultidimensional();  
  
        int array1[][] = { { 1,2,3 }, { 4,5,6 } };  
        int array2[][] = { { 1,2}, {3}, {4,5,6} };  
  
        am.mostra(array1);  
        am.mostra(array2);  
    }  
}
```

Acessa o método mostra disponível no objeto instanciado am, passando como parâmetro o array2



Strings e Caracteres

- String
 - ✓ Seqüência de caracteres tratada como uma unidade única
 - ✓ Pode incluir letras, dígitos e váios caracteres especiais como +, -, *, /, \$ e outros
 - ✓ É um objeto da classe String
 - ✓ Os literais string ou constantes string (tamb'm chamados de objetos String anônimos) são escritos como seqüência de caracteres entre aspas duplas como segue:
 - “Joaquim F. Xavier” Nome
 - “Rua Mamanguape, 233” endereço
 - “Recife – PE” Cidade - estado
 - “(021 81) 3325-5988” telefone

Strings e Caracteres

- String

- ✓ Pode ser atribuído em uma declaração a uma referência de String

- String color = "blue";

Inicializa a referência String color
para referenciar o objeto String "blue"

Strings e Caracteres

- Construtores de String
- ✓ A classe String fornece 9 construtores para inicializar objetos String de diferentes maneiras. Mostraremos apenas os principais no exemplo.

```
import javax.swing.*;

public class ConstrutorString {
    public static void main (String args[]) {

        char charArray[] = {'F','a','c','u','l','d','a','d','e',' ','P','e','r','n','a','m','b','u','c','a','n','a'};

        byte byteArray[] = {(byte) 'F', (byte) 'A', (byte) 'P', (byte) 'E'};

        StringBuffer buffer;

        String s,s1,s2,s3,s4,s5,s6,s7,mostra;

        s = new String("Alô!");

        buffer = new StringBuffer("Bem vindo à FAPE!");
```

Strings e Caracteres

- Construtores de String

✓ A classe String fornece 9 construtores para inicializar objetos String de diferentes maneiras. Mostraremos apenas os principais no exemplo.

// Utiliza os construtores de String

```
s1 = new String();  
s2 = new String( s );  
s3 = new String( charArray );  
s4 = new String( charArray, 10,12);  
s5 = new String( byteArray );  
s6 = new String( byteArray, 2,2);  
s7 = new String( buffer );
```

```
mostra = "s1 = " + s1 +  
        "\ns2 = " + s2 +  
        "\ns3 = " + s3 +  
        "\ns4 = " + s4 +  
        "\ns5 = " + s5 +  
        "\ns6 = " + s6 +  
        "\ns7 = " + s7;
```

```
JOptionPane.showMessageDialog( null, mostra,  
                               "Demonstração de Construtores de String!",  
                               JOptionPane.INFORMATION_MESSAGE );
```

```
System.exit(0);
```

```
}  
}
```

Strings e Caracteres


- Construtores de String

✓ A classe String fornece 9 construtores para inicializar objetos String de diferentes maneiras. Mostraremos apenas os principais no exemplo.

```
import javax.swing.*;
```

```
public class ConstrutorString {  
    public static void main (String args[]) {
```

Definição do array de caracteres
charArray[] e sua inicialização



```
        char charArray[] = {'F','a','c','u','l','d','a','d','e',' ','P','e','r','n','a','m','b','u','c','a','n','a'};
```

```
        byte byteArray[] = {(byte) 'F', (byte) 'A', (byte) 'P', (byte) 'E'};
```

```
        StringBuffer buffer;
```

```
        String s,s1,s2,s3,s4,s5,s6,s7,mostra;
```

```
        s = new String("Alô!");
```

```
        buffer = new StringBuffer("Bem vindo à FAPE!");
```

Strings e Caracteres

- Construtores de String

✓ A classe String fornece 9 construtores para inicializar objetos String de diferentes maneiras. Mostraremos apenas os principais no exemplo.

```
import javax.swing.*;

public class ConstrutorString {
    public static void main (String args[]) {

        char charArray[] = {'F','a','c','u','l','d','a','d','e',' ','P','e','r','n','a','m','b','u','c','a','n','a'};

        byte byteArray[] = {(byte) 'F', (byte) 'A', (byte) 'P', (byte) 'E'};

        StringBuffer buffer;

        String s,s1,s2,s3,s4,s5,s6,s7,mostra;

        s = new String("Alô!");

        buffer = new StringBuffer("Bem vindo à FAPE!");
```

Definição do array de bytes
byteArray[] e sua inicialização

Note o uso de cast para transformar
um caractere em um byte

Strings e Caracteres

- Construtores de String

✓ A classe String fornece 9 construtores para inicializar objetos String de diferentes maneiras. Mostraremos apenas os principais no exemplo.

```
import javax.swing.*;
```

```
public class ConstrutorString {  
    public static void main (String args[]) {
```

```
        char charArray[] = {'F','a','c','u','l','d','a','d','e',' ','P','e','r','n','a','m','b','u','c','a','n','a'};
```

```
        byte byteArray[] = {(byte) 'F', (byte) 'A', (byte) 'P', (byte) 'E'};
```

```
        StringBuffer buffer;
```

```
        String s,s1,s2,s3,s4,s5,s6,s7,mostra;
```

```
        s = new String("Alô!");
```

```
        buffer = new StringBuffer("Bem vindo à FAPE!");
```

Definição do objeto buffer do tipo
StringBuffer



Strings e Caracteres

- Construtores de String

✓ A classe String fornece 9 construtores para inicializar objetos String de diferentes maneiras. Mostraremos apenas os principais no exemplo.

```
import javax.swing.*;

public class ConstrutorString {
    public static void main (String args[]) {

        char charArray[] = {'F','a','c','u','l','d','a','d','e',' ','P','e','r','n','a','m','b','u','c','a','n','a'};

        byte byteArray[] = {(byte) 'F', (byte) 'A', (byte) 'P', (byte) 'E'};

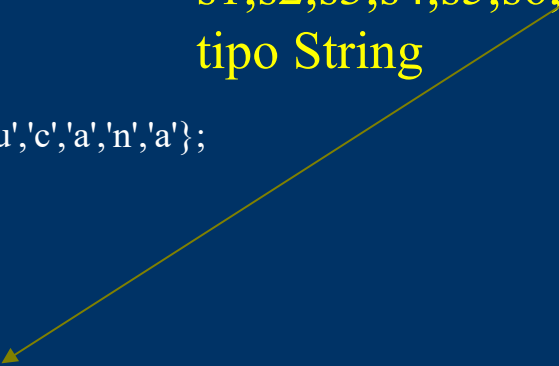
        StringBuffer buffer;

        String s,s1,s2,s3,s4,s5,s6,s7,mostra;

        s = new String("Alô!");

        buffer = new StringBuffer("Bem vindo à FAPE!");
```

Definição dos objetos s,
s1,s2,s3,s4,s5,s6,s7 e mostra do
tipo String



Strings e Caracteres

- Construtores de String

✓ A classe String fornece 9 construtores para inicializar objetos String de diferentes maneiras. Mostraremos apenas os principais no exemplo.

```
import javax.swing.*;
```

```
public class ConstrutorString {  
    public static void main (String args[]) {
```

```
        char charArray[] = {'F','a','c','u','l','d','a','d','e',' ','P','e','r','n','a','m','b','u','c','a','n','a'};
```

```
        byte byteArray[] = {(byte) 'F', (byte) 'A', (byte) 'P', (byte) 'E'};
```

```
        StringBuffer buffer;
```

```
        String s,s1,s2,s3,s4,s5,s6,s7,mostra;
```

```
        s = new String("Alô!");
```

```
        buffer = new StringBuffer("Bem vindo à FAPE!");
```

Inicialização do objeto s do tipo String



Strings e Caracteres

- Construtores de String

✓ A classe String fornece 9 construtores para inicializar objetos String de diferentes maneiras. Mostraremos apenas os principais no exemplo.

```
import javax.swing.*;

public class ConstrutorString {
    public static void main (String args[]) {

        char charArray[] = {'F','a','c','u','l','d','a','d','e',' ','P','e','r','n','a','m','b','u','c','a','n','a'};

        byte byteArray[] = {(byte) 'F', (byte) 'A', (byte) 'P', (byte) 'E'};

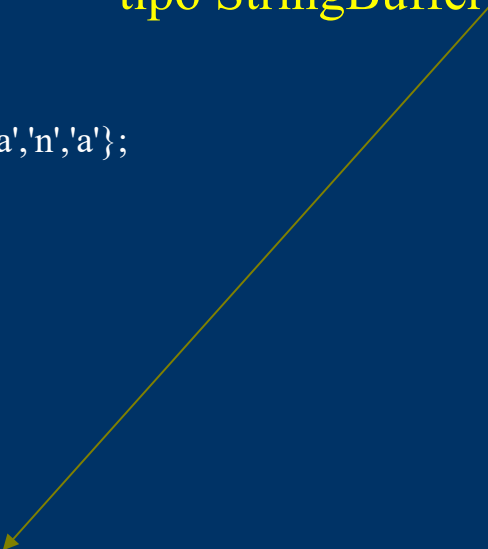
        StringBuffer buffer;

        String s,s1,s2,s3,s4,s5,s6,s7,mostra;

        s = new String("Alô!");

        buffer = new StringBuffer("Bem vindo à FAPE!");
```

Inicialização do objeto buffer do tipo StringBuffer



Strings e Caracteres

- Construtores de String

✓ A classe String fornece 9 construtores para inicializar objetos String de diferentes maneiras. Mostraremos apenas os principais no exemplo.

// Utiliza os construtores de String

```
s1 = new String();  
s2 = new String( s );  
s3 = new String( charArray );  
s4 = new String( charArray, 10,12);  
s5 = new String( byteArray );  
s6 = new String( byteArray, 2,2);  
s7 = new String( buffer );
```

Diferentes construtores para
objetos do tipo String



```
mostra = "s1 = " + s1 +  
        "\ns2 = " + s2 +  
        "\ns3 = " + s3 +  
        "\ns4 = " + s4 +  
        "\ns5 = " + s5 +  
        "\ns6 = " + s6 +  
        "\ns7 = " + s7;
```

```
JOptionPane.showMessageDialog( null, mostra,  
                               "Demonstração de Construtores de String!",  
                               JOptionPane.INFORMATION_MESSAGE );
```

```
System.exit(0);
```

```
}  
}
```

Strings e Caracteres

- Construtores de String

✓ A classe String fornece 9 construtores para inicializar objetos String de diferentes maneiras. Mostraremos apenas os principais no exemplo.

// Utiliza os construtores de String

```
s1 = new String();  
s2 = new String( s );  
s3 = new String( charArray );  
s4 = new String( charArray, 10,12);  
s5 = new String( byteArray );  
s6 = new String( byteArray, 2,2);  
s7 = new String( buffer );
```

Cria String mostra a partir do valor das strings criadas



```
mostra = "s1 = " + s1 +  
        "\ns2 = " + s2 +  
        "\ns3 = " + s3 +  
        "\ns4 = " + s4 +  
        "\ns5 = " + s5 +  
        "\ns6 = " + s6 +  
        "\ns7 = " + s7;
```

```
JOptionPane.showMessageDialog( null, mostra,  
                              "Demonstração de Construtores de String!",  
                              JOptionPane.INFORMATION_MESSAGE );
```

```
System.exit(0);
```

```
}  
}
```

Strings e Caracteres

- Construtores de String

✓ A classe String fornece 9 construtores para inicializar objetos String de diferentes maneiras. Mostraremos apenas os principais no exemplo.

// Utiliza os construtores de String

```
s1 = new String();  
s2 = new String( s );  
s3 = new String( charArray );  
s4 = new String( charArray, 10,12);  
s5 = new String( byteArray );  
s6 = new String( byteArray, 2,2);  
s7 = new String( buffer );
```

```
mostra = "s1 = " + s1 +  
        "\ns2 = " + s2 +  
        "\ns3 = " + s3 +  
        "\ns4 = " + s4 +  
        "\ns5 = " + s5 +  
        "\ns6 = " + s6 +  
        "\ns7 = " + s7;
```

Mostra através do método estático
showMessageDialog da classe
JOptionPane

```
JOptionPane.showMessageDialog( null, mostra,  
                             "Demonstração de Construtores de String!",  
                             JOptionPane.INFORMATION_MESSAGE );
```

```
System.exit(0);
```

```
}  
}
```

Strings e Caracteres

- Métodos String length, charAt e getChars
 - ✓ String length **determina o comprimento de um String**
 - **String s1 = “FAPE”;**
 - **s1.length();**
 - ♦ **Retorna 4**
 - ✓ charAt **obtém o caractere em uma localização específica em um String**
 - **char pedaco = s1.charAt(3);**
 - ♦ **Retorna o caractere E**
 - ✓ getChars **obtém o conjunto inteiro de caracteres em um String**
 - **char charArray[];**
 - **s1.getChars(0, 2, charArray, 0);**
 - ♦ **Atribui a charArray os dois primeiros caracteres da string s1, ou seja, FA**
- ✓ Exercício: Explore estas funcionalidades de manipulação de Strings em um programa JAVA

Strings e Caracteres

- Alterando o exemplo anterior para demonstrar os métodos String length, charAt e getChars

```
import javax.swing.*;

public class ManipulacaoString {
    public static void main (String args[]) {

        char charArray[] = {'F','a','c','u','l','d','a','d','e',' ','P','e','r','n','a','m','b','u','c','a','n','a'};

        byte byteArray[] = {(byte) 'F', (byte) 'A', (byte) 'P', (byte) 'E'};

        StringBuffer buffer;

        String s,s1,s2,s3,s4,s5,s6,s7,mostra;

        s = new String("Alô!");

        buffer = new StringBuffer("Bem vindo à FAPE!");

        // Utiliza os construtores de String
        s1 = new String();
        s2 = new String( s );
        s3 = new String( charArray );
        s4 = new String( charArray, 10,12);
```

Strings e Caracteres

- Alterando o exemplo anterior para demonstrar os métodos String length, charAt e getChars

```
s5 = new String( byteArray );  
s6 = new String( byteArray, 2,2);  
s7 = new String( buffer );
```

```
char caracteres[];  
caracteres = new char[5];
```

```
s7.getChars(0,3,caracteres,0);
```

```
System.out.println(caracteres);
```

```
System.out.println("s1 possui tamanho " + s1.length());  
System.out.println("s2 possui tamanho " + s2.length());  
System.out.println("s3 possui tamanho " + s3.length());  
System.out.println("s4 possui tamanho " + s4.length());  
System.out.println("s5 possui tamanho " + s5.length());  
System.out.println("s6 possui tamanho " + s6.length());  
System.out.println("s7 possui tamanho " + s7.length());
```

```
System.out.println();
```

Strings e Caracteres

- Alterando o exemplo anterior para demonstrar os métodos String length, charAt e getChars

```
System.out.println("a primeira letra de s2 é " + s2.charAt(0));
System.out.println("a primeira letra de s3 é " + s3.charAt(0));
System.out.println("a primeira letra de s3 é " + s3.charAt(0));
System.out.println("a primeira letra de s4 é " + s4.charAt(0));
System.out.println("a primeira letra de s5 é " + s5.charAt(0));
System.out.println("a primeira letra de s6 é " + s6.charAt(0));
System.out.println("a primeira letra de s7 é " + s7.charAt(0));
```

```
mostra = "s1 = " + s1 +
        "\ns2 = " + s2 +
        "\ns3 = " + s3 +
        "\ns4 = " + s4 +
        "\ns5 = " + s5 +
        "\ns6 = " + s6 +
        "\ns7 = " + s7 +
        "\ncaracteres = " + new String(caracteres);
```

```
JOptionPane.showMessageDialog( null, mostra,
                                "Demonstração de Construtores de String!",
                                JOptionPane.INFORMATION_MESSAGE );
```

```
System.exit(0);
```

```
}
}
```

Strings e Caracteres

- Comparação de Strings
 - ✓ Métodos String equals, equalsIgnoreCase, compareTo e regionMatches
 - equals
 - ♦ Realiza a comparação de duas strings levando em conta a diferença entre maiúsculas e minúsculas, retornando true caso sejam iguais
 - equalsIgnoreCase
 - ♦ Realiza a comparação de duas strings considerando a igualdade entre maiúsculas e minúsculas, retornando true caso sejam iguais

Strings e Caracteres

- Comparação de Strings
 - ✓ Métodos String equals, equalsIgnoreCase, compareTo e regionMatches
 - compareTo
 - ◆ Retorna 0 se os strings forem iguais
 - ◆ Retorna um número negativo se o string que invoca compareTo for menor que o string que é passado como argumento
 - ◆ Retorna um número positivo se o string que invoca compareTo for maior que o string que é passado como argumento
 - regionMatches
 - ◆ Compara partes de dois objetos String quanto à igualdade

Strings e Caracteres

- Comparação de Strings

- ✓ Método String equals

```
import javax.swing.JOptionPane;
```

```
public class ComparandoStrings {  
    public static void main ( String args[] ) {  
        String s1, s2, s3, s4, saida;  
  
        s1 = new String("Alô!");  
        s2 = new String("Tchau");  
        s3 = new String("Feliz Aniversário");  
        s4 = new String("feliz aniversário");  
  
        saida = "s1 = " + s1 + "\ns2 = " + s2 + "\ns3 = " +  
            s3 + "\ns4 = " + s4 + "\n\n";  
  
        // teste de igualdade usando equals  
  
        if ( s1.equals("Alô!") )  
            saida += "s1 é igual a \"hello\" quando se usa " +  
                "método equals da classe String\n";  
        else  
            saida += "s1 não é igual a \"hello\" quando " +  
                "se usa operador == \n";  
    }  
}
```

Strings e Caracteres

- Comparação de Strings
- ✓ Método String equals

```
// teste de igualdade usando ==
```

```
if ( s1 == "Alô!" )
```

```
    saida += "s1 igual a \"hello\\n\"";
```

```
else
```

```
    saida += "s1 não é igual a \"hello\\n\"";
```

```
JOptionPane.showMessageDialog(null, saida,  
    "Demonstrando comparação de Strings",  
    JOptionPane.INFORMATION_MESSAGE );
```

```
System.exit( 0 );
```

```
}
```

```
}
```

Strings e Caracteres

- Comparação de Strings

- ✓ Método String equals **Explicação**

```
import javax.swing.JOptionPane;
```

```
public class ComparandoStrings {  
    public static void main ( String args[] ) {  
        String s1, s2, s3, s4, saida;  
  
        s1 = new String("Alô!");  
        s2 = new String("Tchau");  
        s3 = new String("Feliz Aniversário");  
        s4 = new String("feliz aniversário");  
  
        saida = "s1 = " + s1 + "\ns2 = " + s2 + "\ns3 = " +  
            s3 + "\ns4 = " + s4 + "\n\n";
```

```
// teste de igualdade usando equals
```

```
if ( s1.equals("Alô!") )  
    saida += "s1 é igual a \"hello\" quando se usa " +  
            "método equals da classe String\n";  
else  
    saida += "s1 não é igual a \"hello\" quando " +  
            "se usa operador == \n";
```

Forma correta de se comparar duas strings

Strings e Caracteres

- Comparação de Strings
- ✓ Método String equals **Explicação**

```
// teste de igualdade usando ==
```

```
if( s1 == "Alô!" )  
    saida += "s1 igual a \"hello\"\\n";  
else  
    saida += "s1 não é igual a \"hello\"\\n";
```

Forma incorreta de comparação de strings. Na verdade se está comparando a referência dos objetos

```
JOptionPane.showMessageDialog(null, saida,  
    "Demonstrando comparação de Strings",  
    JOptionPane.INFORMATION_MESSAGE );
```

```
System.exit( 0 );
```

```
}
```

```
}
```

Strings e Caracteres

- Comparação de Strings
- ✓ Método String compareTo

```
import javax.swing.JOptionPane;

public class ComparandoStrings2 {
    public static void main ( String args[] ) {
        String s1, s2, s3, s4, saida;

        s1 = new String("Alô!");
        s2 = new String("Tchau");
        s3 = new String("Feliz Aniversário");
        s4 = new String("feliz aniversário");

        saida = "s1 = " + s1 + "\ns2 = " + s2 + "\ns3 = " +
            s3 + "\ns4 = " + s4 + "\n\n";

        // testa compareTo

        saida += "\ns1.compareTo( s2 ) é " + s1.compareTo(s2) +
            "\ns2.compareTo( s1 ) é " + s2.compareTo(s1) +
            "\ns1.compareTo( s1 ) é " + s1.compareTo(s1) +
            "\ns3.compareTo( s4 ) é " + s3.compareTo(s4) +
            "\ns4.compareTo( s3 ) é " + s4.compareTo(s3) +
            "\n\n";
    }
}
```

Strings e Caracteres

- Comparação de Strings
- ✓ Método String compareTo

```
JOptionPane.showMessageDialog(null, saida,  
    "Demonstrando comparação de Strings",  
    JOptionPane.INFORMATION_MESSAGE );
```

```
System.exit( 0 );
```

```
}
```

```
}
```

Strings e Caracteres

- Comparação de Strings
- ✓ Método String compareTo **Explicação**

```
import javax.swing.JOptionPane;

public class ComparandoStrings2 {
    public static void main ( String args[] ) {
        String s1, s2, s3, s4, saida;

        s1 = new String("Alô!");
        s2 = new String("Tchau");
        s3 = new String("Feliz Aniversário");
        s4 = new String("feliz aniversário");

        saida = "s1 = " + s1 + "\ns2 = " + s2 + "\ns3 = " +
            s3 + "\ns4 = " + s4 + "\n\n";

        // testa compareTo

        saida += "\ns1.compareTo( s2 ) é " + s1.compareTo(s2) +
            "\ns2.compareTo( s1 ) é " + s2.compareTo(s1) +
            "\ns1.compareTo( s1 ) é " + s1.compareTo(s1) +
            "\ns3.compareTo( s4 ) é " + s3.compareTo(s4) +
            "\ns4.compareTo( s3 ) é " + s4.compareTo(s3) +
            "\n\n";
    }
}
```

Compara as strings s1 com s2,
retornando o valor 0 caso sejam iguais;
retornando um número negativo se o
s1 for menor que s2;
retornando um número positivo se s1
for maior que s2

Strings e Caracteres

- Localizando caracteres e substrings em Strings
 - ✓ IndexOf
 - Localiza a posição de caracteres ou palavras em uma String, considerando-a por inteiro ou apenas uma parte
 - ✓ lastIndexOf
 - Procura a última ocorrência de um caractere ou palavra (substring) em um String
-
-

Strings e Caracteres

- Localizando caracteres e substrings em Strings

```
import javax.swing.JOptionPane;

public class PesquisandoStrings {
    public static void main ( String args[] ) {
        String frase = "Continue estudando na FAPE!";
        String saida;

        saida = "A frase é " + frase;
        saida += "\n'e' está localizado no índice " + frase.indexOf('e');
        saida += "\nO próximo 'e' a partir do índice 9 está localizado no índice " + frase.indexOf('e',9);
        saida += "\no último 'a' está localizado no índice " + frase.lastIndexOf('a');

        JOptionPane.showMessageDialog(null, saida,
            "Demonstrando comparação de Strings",
            JOptionPane.INFORMATION_MESSAGE );

        System.exit( 0 );
    }
}
```

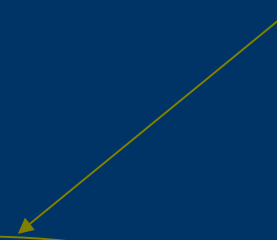
Strings e Caracteres

- Localizando caracteres e substrings em Strings **Explicação**

```
import javax.swing.JOptionPane;
```

```
public class PesquisandoStrings {  
    public static void main ( String args[] ) {  
        String frase = "Continue estudando na FAPE!";  
        String saida;  
  
        saida = "A frase é " + frase;  
        saida += "\n'e' está localizado no índice " + frase.indexOf('e');  
        saida += "\nO próximo 'e' a partir do índice 9 está localizado no índice " + frase.indexOf('e',9);  
        saida += "\no último 'a' está localizado no índice " + frase.lastIndexOf('a');  
  
        JOptionPane.showMessageDialog(null, saida,  
            "Demonstrando comparação de Strings",  
            JOptionPane.INFORMATION_MESSAGE );  
  
        System.exit( 0 );  
    }  
}
```

Retorna o valor do índice onde se localiza o caractere 'e' i.e. 7



Strings e Caracteres

- Localizando caracteres e substrings em Strings **Explicação**

```
import javax.swing.JOptionPane;
```

```
public class PesquisandoStrings {
```

```
    public static void main ( String args[] ) {
```

```
        String frase = "Continue estudando na FAPE!";
```

```
        String saida;
```

```
        saida = "A frase é " + frase;
```

```
        saida += "\n'e' está localizado no índice " + frase.indexOf('e');
```

```
        saida += "\nO próximo 'e' a partir do índice 9 está localizado no índice " + frase.indexOf('e',9);
```

```
        saida += "\no último 'a' está localizado no índice " + frase.lastIndexOf('a');
```

```
        JOptionPane.showMessageDialog(null, saida,
```

```
            "Demonstrando comparação de Strings",
```

```
            JOptionPane.INFORMATION_MESSAGE );
```

```
        System.exit( 0 );
```

```
    }
```

```
}
```

Retorna o valor do índice onde se localiza o próximo caractere 'e' a partir da posição 9 i.e. 9



Strings e Caracteres

- Localizando caracteres e substrings em Strings **Explicação**

```
import javax.swing.JOptionPane;
```

```
public class PesquisandoStrings {  
    public static void main ( String args[] ) {  
        String frase = "Continue estudando na FAPE!";  
        String saida;  
  
        saida = "A frase é " + frase;  
        saida += "\n'e' está localizado no índice " + frase.indexOf('e');  
        saida += "\nO próximo 'e' a partir do índice 9 está localizado no índice " + frase.indexOf('e',9);  
        saida += "\no último 'a' está localizado no índice " + frase.lastIndexOf('a');  
  
        JOptionPane.showMessageDialog(null, saida,  
            "Demonstrando comparação de Strings",  
            JOptionPane.INFORMATION_MESSAGE );  
  
        System.exit( 0 );  
    }  
}
```

Retorna o valor do índice onde se localiza o último caractere 'a' na frase i.e 20

Strings e Caracteres

- Extraíndo substrings a partir de Strings
- ✓ Permite que um novo objeto String seja criado, copiando parte de um objeto String já existente

Strings e Caracteres

- Extraíndo substrings a partir de Strings

```
import javax.swing.JOptionPane;
```

```
public class CriandoSubStrings {  
    public static void main ( String args[] ) {  
        String frase = "Continue estudando na FAPE!";  
        String saida, pedaco, pedaco1;  
  
        pedaco = frase.substring(8);  
        pedaco1 = frase.substring(9,19);  
        System.out.println(pedaco1);  
        saida = "A frase é " + frase;  
        saida += "\nA substring a partir da posição 8 é " + pedaco;  
        saida += "\nA substring a partir da posição 8, pegando apenas 9 caracteres é " + pedaco1;  
  
        JOptionPane.showMessageDialog(null, saida,  
            "Demonstrando comparação de Strings",  
            JOptionPane.INFORMATION_MESSAGE );  
  
        System.exit( 0 );  
    }  
}
```


Strings e Caracteres

• Extraíndo substrings a partir de Strings Explicação

```
import javax.swing.JOptionPane;
```

```
public class CriandoSubStrings {  
    public static void main ( String args[] ) {  
        String frase = "Continue estudando na FAPE!";  
        String saida, pedaco, pedaco1;  
  
        pedaco = frase.substring(8);  
        pedaco1 = frase.substring(9,19);  
        System.out.println(pedaco1);  
        saida = "A frase é " + frase;  
        saida += "\nA substring a partir da posição 8 é " + pedaco;  
        saida += "\nA substring a partir da posição 8, pegando apenas 9 caracteres é " + pedaco1;  
  
        JOptionPane.showMessageDialog(null, saida,  
            "Demonstrando comparação de Strings",  
            JOptionPane.INFORMATION_MESSAGE );  
  
        System.exit( 0 );  
    }  
}
```

Monta uma nova String a partir da posição 8 da String armazenada na variável frase i.e estudando na FAPE!



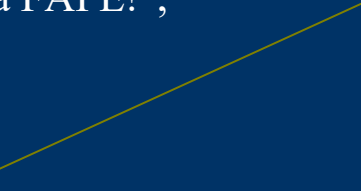
Strings e Caracteres

• Extraíndo substrings a partir de Strings Explicação

```
import javax.swing.JOptionPane;
```

```
public class CriandoSubStrings {  
    public static void main ( String args[] ) {  
        String frase = "Continue estudando na FAPE!";  
        String saida, pedaco, pedaco1;  
  
        pedaco = frase.substring(8);  
        pedaco1 = frase.substring(9,19);  
        System.out.println(pedaco1);  
        saida = "A frase é " + frase;  
        saida += "\nA substring a partir da posição 8 é " + pedaco;  
        saida += "\nA substring a partir da posição 8, pegando apenas 9 caracteres é " + pedaco1;  
  
        JOptionPane.showMessageDialog(null, saida,  
            "Demonstrando comparação de Strings",  
            JOptionPane.INFORMATION_MESSAGE );  
  
        System.exit( 0 );  
    }  
}
```

Monta uma nova String a partir da posição 9 até a posição 19 da String armazenada na variável frase i.e estudando



Arquivos e Fluxos

- Arquivos Seqüenciais

- ✓ Escrita - Classe GravaFuncionario

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class GravaFuncionario {
    public static void main (String args[]) {
        String nome, fone, nomeArquivo;
        ObjectOutputStream output=null;

        nomeArquivo = "/tmp/dados.dat";

        // abre arquivo para escrita
        try {
            output = new ObjectOutputStream(
                new FileOutputStream( nomeArquivo ));

        } catch (IOException e ) {

            System.out.println("Erro ao abrir arquivo!");

        }
    }
}
```

Arquivos e Fluxos

- Arquivos Seqüenciais
- ✓ Escrita - Classe GravaFuncionario

```
do {  
  
    System.out.println("Informe o nome do funcionário ou 'sai' para sair ");  
    nome = Util.readStr();  
  
    System.out.println("Informe o fone do funcionário ");  
    fone = Util.readStr();  
  
    Funcionario f = new Funcionario();  
    f.setNome(nome);  
    f.setFone(fone);  
  
    if (!f.getNome().equals("sai")) {  
        // insere registro  
        try {  
  
            output.writeObject(f);  
            output.flush();  
  
        } catch (Exception e) {  
            System.out.println("Erro!");  
        }  
    }  
}
```

Arquivos e Fluxos

- Arquivos Seqüenciais
 - ✓ Escrita - Classe GravaFuncionario

```
    } while (!nome.equals("sai") );  
  
    // fecha arquivo  
  
    try {  
        output.close();  
  
    } catch(IOException ex) {  
        System.out.println("Erro ao fechar arquivo!");  
  
    }  
  
}
```

Arquivos e Fluxos

- Arquivos Seqüenciais
 - ✓ Escrita - Classe Funcionario

```
import java.io.Serializable;
```

```
public class Funcionario implements Serializable {
```

```
    private String nome;
```

```
    private String fone;
```

```
    public String getNome() {
```

```
        return nome;
```

```
    }
```

```
    public String getFone() {
```

```
        return fone;
```

```
    }
```

```
    public void setNome(String n) {
```

```
        nome = n;
```

```
    }
```

```
    public void setFone(String f) {
```

```
        fone = f;
```

```
    }
```

```
}
```

Arquivos e Fluxos

- Arquivos Seqüenciais

- ✓ Escrita - Classe GravaFuncionario

Explicação

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
public class GravaFuncionario {
    public static void main (String args[]) {
        String nome, fone, nomeArquivo;
        ObjectOutputStream output=null;

        nomeArquivo = "/tmp/dados.dat";

        // abre arquivo para escrita
        try {
            output = new ObjectOutputStream(
                new FileOutputStream( nomeArquivo ));

        } catch (IOException e ) {

            System.out.println("Erro ao abrir arquivo!");

        }
    }
}
```

Objeto output do tipo ObjectOutputStream usado para gravar objetos da classe funcionário no banco

Arquivos e Fluxos

- Arquivos Seqüenciais

- ✓ Escrita - Classe GravaFuncionario

Explicação

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
public class GravaFuncionario {
    public static void main (String args[]) {
        String nome, fone, nomeArquivo;
        ObjectOutputStream output=null;
```

```
        nomeArquivo = "/tmp/dados.dat";
```

Caminho e nome do arquivo a ser criado



```
// abre arquivo para escrita
```

```
try {
    output = new ObjectOutputStream(
        new FileOutputStream( nomeArquivo ));
```

```
} catch (IOException e ) {
```

```
    System.out.println("Erro ao abrir arquivo!");
```

```
}
```

Arquivos e Fluxos

- Arquivos Seqüenciais

- ✓ Escrita - Classe GravaFuncionario

Explicação

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
public class GravaFuncionario {
    public static void main (String args[]) {
        String nome, fone, nomeArquivo;
        ObjectOutputStream output=null;
```

```
        nomeArquivo = "/tmp/dados.dat";
```

Abre arquivo para escrita e o associa ao objeto output criado



```
// abre arquivo para escrita
try {
    output = new ObjectOutputStream(
        new FileOutputStream( nomeArquivo ));
} catch (IOException e ) {

    System.out.println("Erro ao abrir arquivo!");
}
```


Arquivos e Fluxos


- Arquivos Seqüenciais

- ✓ Escrita - Classe GravaFuncionario

Explicação

```
do {  
  
    System.out.println("Informe o nome do funcionário ou 'sai' para sair ");  
    nome = Util.readStr();  
  
    System.out.println("Informe o fone do funcionário ");  
    fone = Util.readStr();  
  
    Funcionario f = new Funcionario();  
    f.setNome(nome);  
    f.setFone(fone);  
  
    if (!f.getNome().equals("sai")) {  
        // insere registro  
        try {  
  
            output.writeObject(f);  
            output.flush();  
  
        } catch (Exception e) {  
            System.out.println("Erro!");  
        }  
    }  
}
```

Enquanto não digitar sai para nome, continua inserindo objetos da classe Funcionario no arquivo aberto



Arquivos e Fluxos

- Arquivos Seqüenciais

- ✓ Escrita - Classe GravaFuncionario

Explicação

```
} while (!nome.equals("sai") );  
  
// fecha arquivo  
try {  
    output.close();  
  
} catch(IOException ex) {  
    System.out.println("Erro ao fechar arquivo!");  
  
}  
  
}
```

Fecha o arquivo



Arquivos e Fluxos

- Arquivos Seqüenciais

- ✓ Escrita - Classe Funcionario

Explicação

```
import java.io.Serializable;
```

```
public class Funcionario implements Serializable {  
    private String nome;  
    private String fone;
```

Classe Funcionario implementa a interface **Serializable**, permitindo que objetos complexos sejam gravados no arquivo

```
    public String getNome() {  
        return nome;  
    }
```

```
    public String getFone() {  
        return fone;  
    }
```

```
    public void setNome(String n) {  
        nome = n;  
    }
```

```
    public void setFone(String f) {  
        fone = f;  
    }  
}
```

Arquivos e Fluxos

- Arquivos Seqüenciais

- ✓ Escrita - Classe Funcionario

Explicação

```
import java.io.Serializable;
```

```
public class Funcionario implements Serializable {  
    private String nome;  
    private String fone;
```

Classe Funcionario implementa a interface **Serializable**, permitindo que objetos complexos sejam gravados no arquivo

```
    public String getNome() {  
        return nome;  
    }
```

```
    public String getFone() {  
        return fone;  
    }
```

```
    public void setNome(String n) {  
        nome = n;  
    }
```

```
    public void setFone(String f) {  
        fone = f;  
    }  
}
```

Arquivos e Fluxos

- Arquivos Seqüenciais

- ✓ Leitura

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LeFuncionario {
    public static void main (String args[]) {
        String nome, fone, nomeArquivo;
        ObjectInputStream input=null;

        nomeArquivo = "/tmp/dados.dat";

        // abre arquivo para leitura
        try {
            input = new ObjectInputStream(
                new FileInputStream( nomeArquivo ));

        } catch (IOException e ) {

            System.out.println("Erro ao abrir arquivo!");

        }
    }
}
```

Arquivos e Fluxos

- Arquivos Seqüenciais

- ✓ Leitura

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
public class LeFuncionario {
    public static void main (String args[]) {
        String nome, fone, nomeArquivo;
        ObjectInputStream input=null;

        nomeArquivo = "/tmp/dados.dat";

        // abre arquivo para leitura
        try {
            input = new ObjectInputStream(
                new FileInputStream( nomeArquivo ));

        } catch (IOException e ) {

            System.out.println("Erro ao abrir arquivo!");

        }
    }
}
```

Objeto input da classe ObjectInputStream
utilizado para ler do arquivo



Arquivos e Fluxos

- Arquivos Seqüenciais
 - ✓ Leitura

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

Abre o arquivo para leitura



```
public class LeFuncionario {
    public static void main (String args[]) {
        String nome, fone, nomeArquivo;
        ObjectInputStream input=null;

        nomeArquivo = "/tmp/dados.dat";
```

```
// abre arquivo para leitura
try {
    input = new ObjectInputStream(
        new FileInputStream( nomeArquivo ));
} catch (IOException e ) {

    System.out.println("Erro ao abrir arquivo!");
}
```

Arquivos e Fluxos

- Arquivos Seqüenciais
- ✓ Leitura

Lê o arquivo enquanto existirem registros a serem lidos



```
// Lê os registros do arquivo  
Funcionario f=null;
```

```
Boolean fim = false;
```

```
do {  
  
    try {  
        f = (Funcionario)input.readObject();  
  
        System.out.println(f.getNome() + " " + f.getFone());  
  
    } catch (ClassNotFoundException cnfex) {  
        System.out.println("Classe não encontrada!");  
        fim = true;  
  
    } catch (EOFException eofex) {  
        System.out.println("Fim do arquivo!");  
        fim = true;  
  
    }  
  
}
```


Arquivos e Fluxos

- Arquivos Seqüenciais
 - ✓ Leitura

```
catch (IOException ioex) {  
    System.out.println("Erro durante a leitura do arquivo!");  
    fim = true;
```

```
}
```

Fecha o arquivo



```
} while (!fim);
```

```
// fecha arquivo
```

```
try {  
    input.close();
```

```
} catch(IOException ex) {  
    System.out.println("Erro ao fechar arquivo!");
```

```
}
```

```
}
```

```
}
```

Acesso a Banco de Dados

- Atividades associadas ao acesso a Banco de Dados
 - ✓ Abrir conexão com o Banco de Dados
 - ✓ Manipular o Banco de Dados
 - ➔ Alterar a estrutura do Banco de Dados através da DDL – Data Definition Language
 - ♦ Inserir tabelas
 - ♦ Alterar tabelas
 - ♦ Excluir tabelas
 - ♦ Definir Visões
 - ♦ Alterar Visões
 - ♦ Excluir Visões
 - ♦ Definir índices

Acesso a Banco de Dados

- Atividades associadas ao acesso a Banco de Dados
 - ✓ Manipular o Banco de Dados
 - ➔ Manipular os dados no Banco de Dados através da DML – Data Manipulation Language
 - ♦ Inserir registros
 - ♦ Alterar registros
 - ♦ Excluir registros
 - ♦ Consultar os dados armazenados no Banco de Dados
 - ✓ Fechar Conexão ao Banco de Dados

Acesso a Banco de Dados

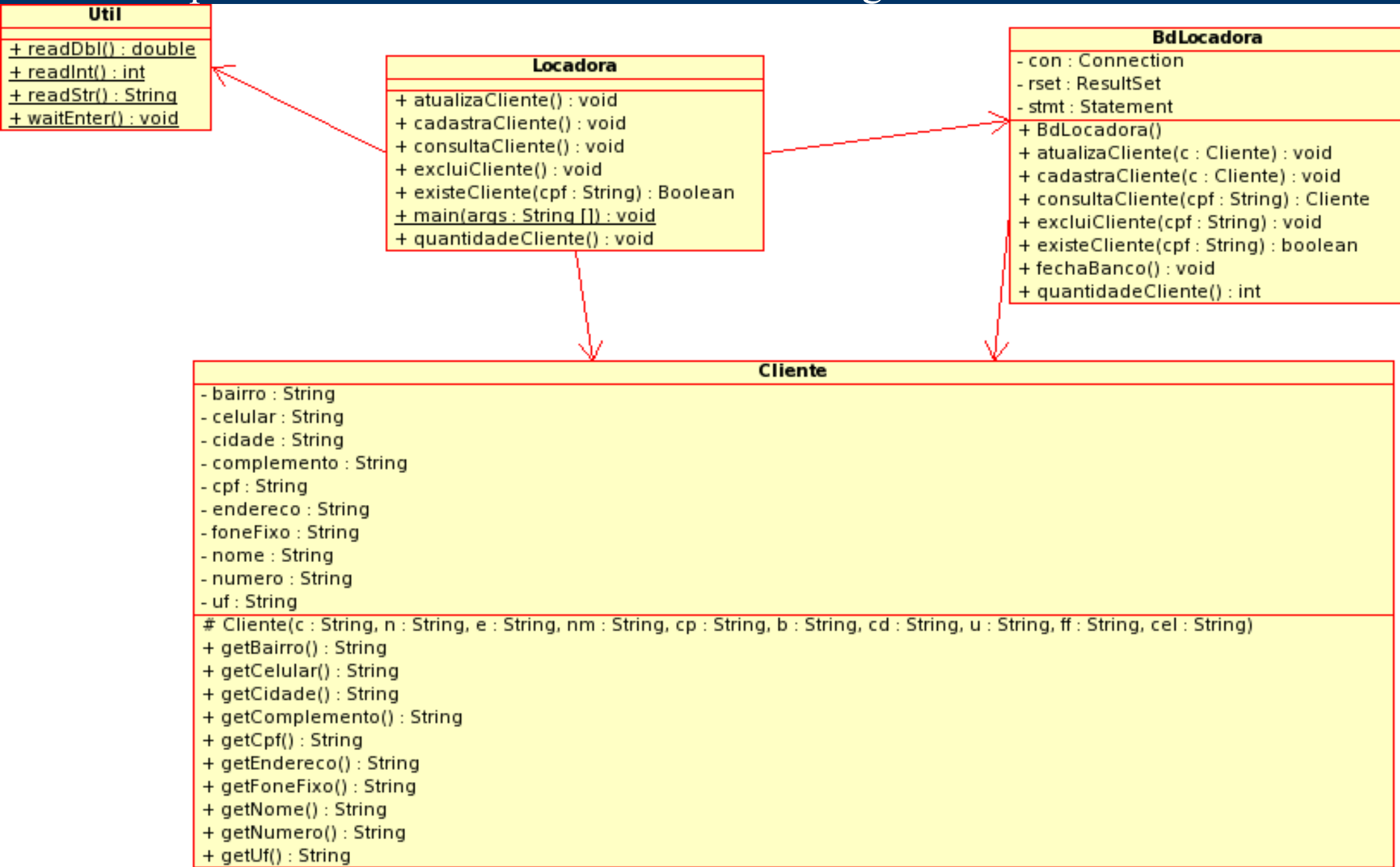
- Structured Query Language (SQL)
 - ✓ Clássica linguagem de consulta a banco de dados
 - Padrões 89, 92, SQL3
 - Possui recursos tanto de
 - DML
 - ♦ SELECT
 - ♦ INSERT
 - ♦ UPDATE
 - ♦ DELETE
 - DDL
 - ♦ CREATE TABLE
 - ♦ ALTER TABLE
 - ♦ DROP TABLE
 - ♦ ...
 - ✓ Opção \h do PostgreSQL mostra ajuda para todos os comandos SQL tanto DML quanto DDL
-
-

Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA
 - ✓ Ambiente
 - Servidor com Sistema Operacional Linux
 - Banco de Dados PostgreSQL 7.4
 - ✓ Classes da Aplicação
 - Locadora.java
 - BdLocadora.java
 - Cliente.java
 - Util.java

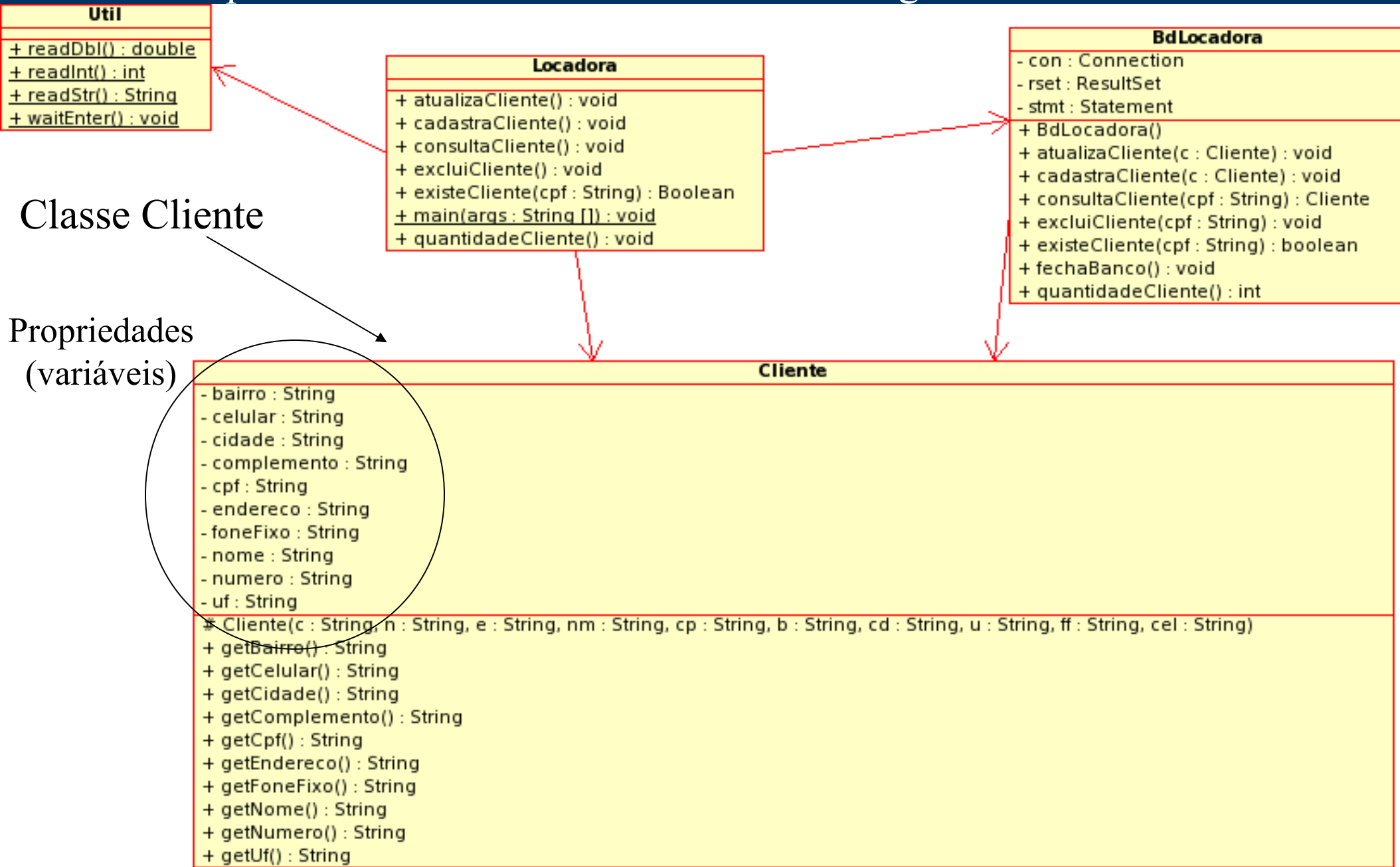
Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



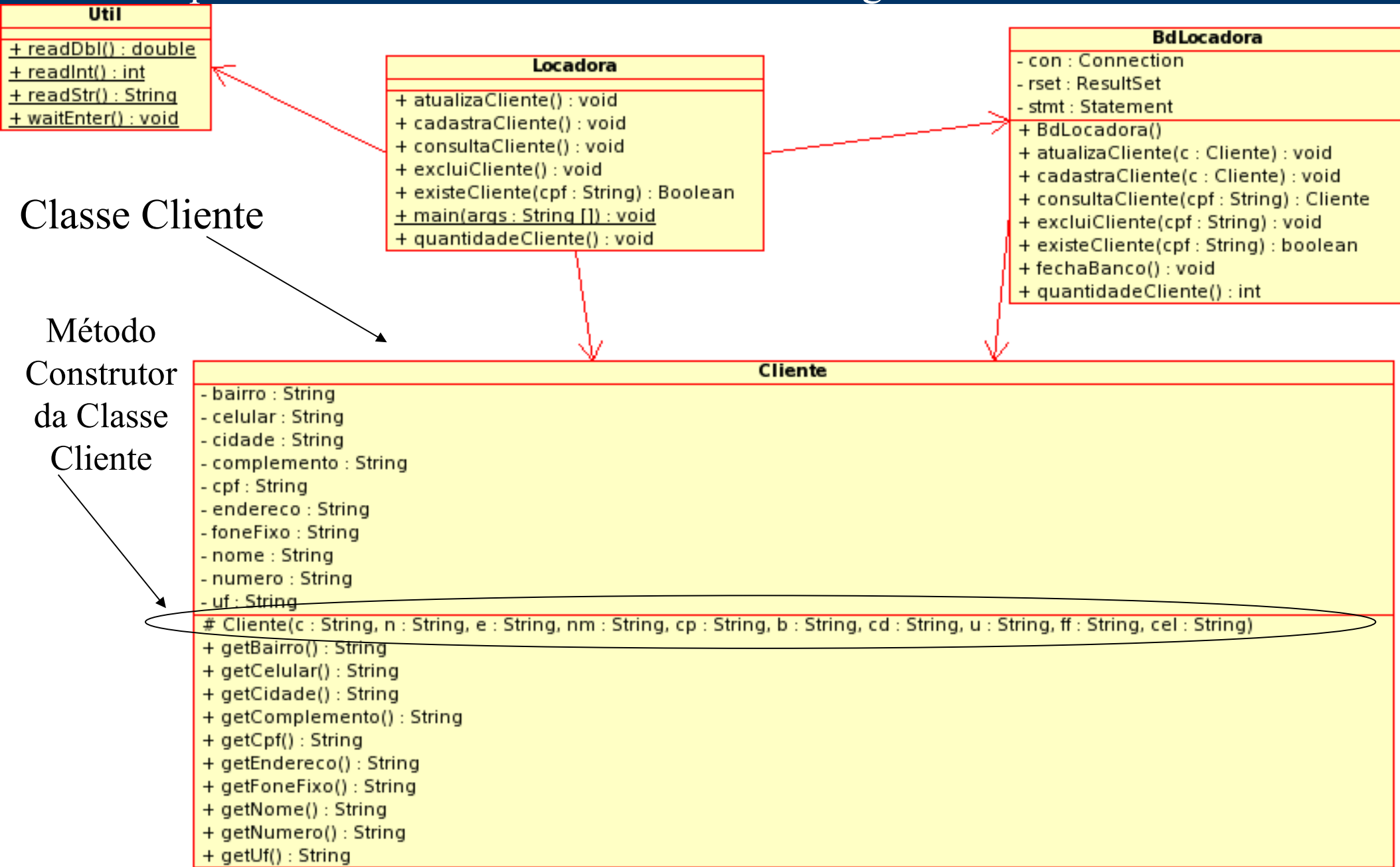
Acesso a Banco de Dados

Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



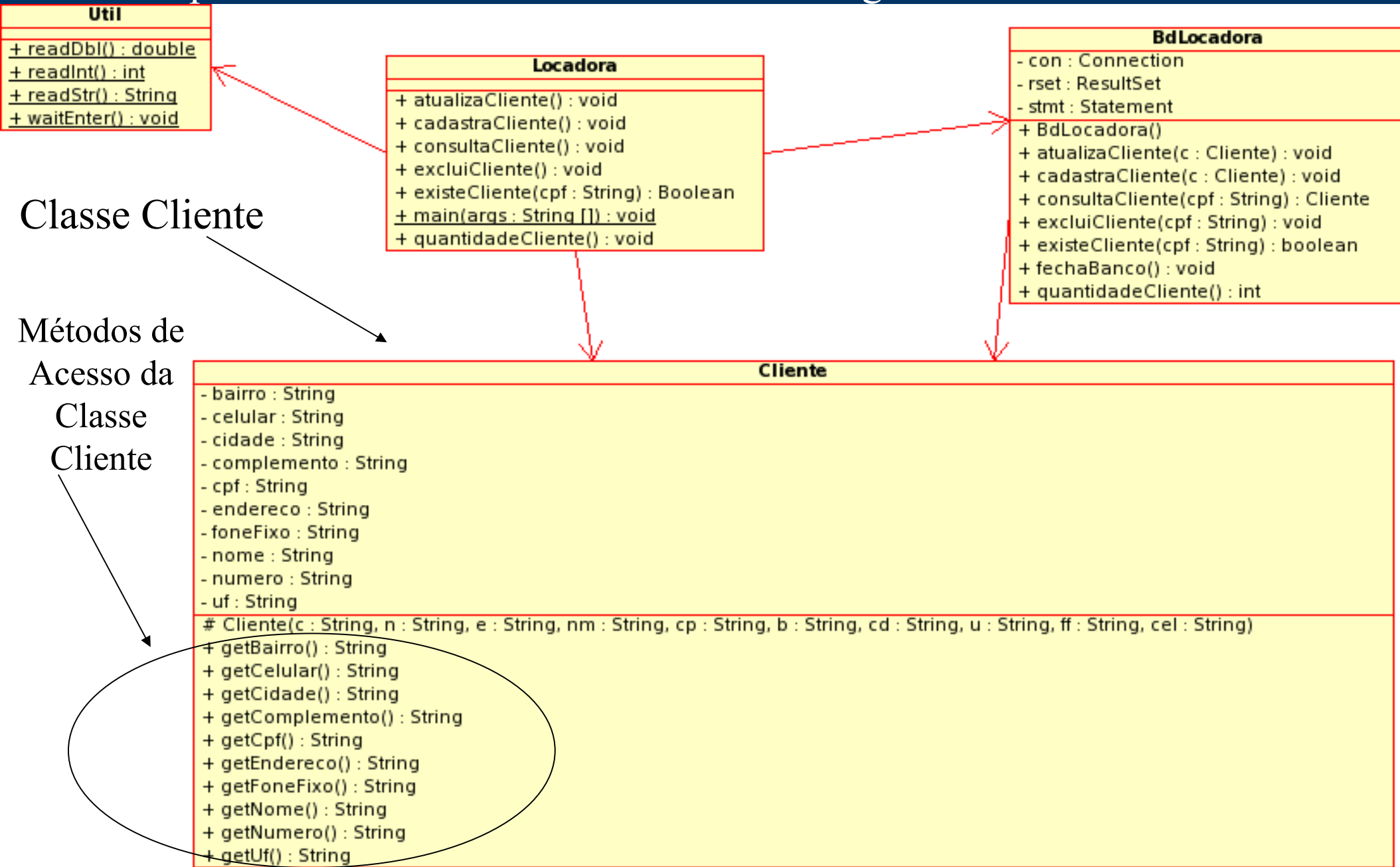
Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



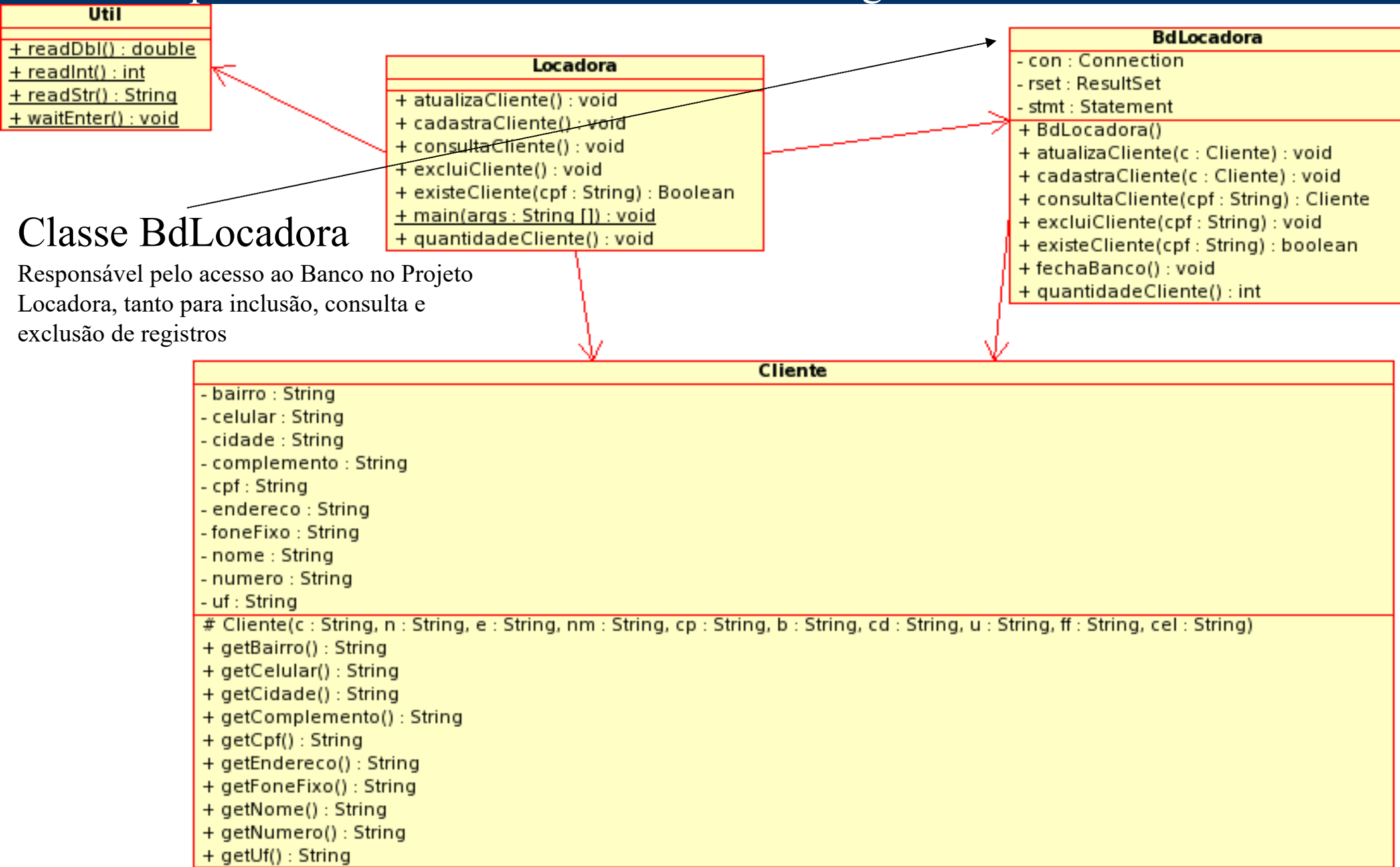
Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



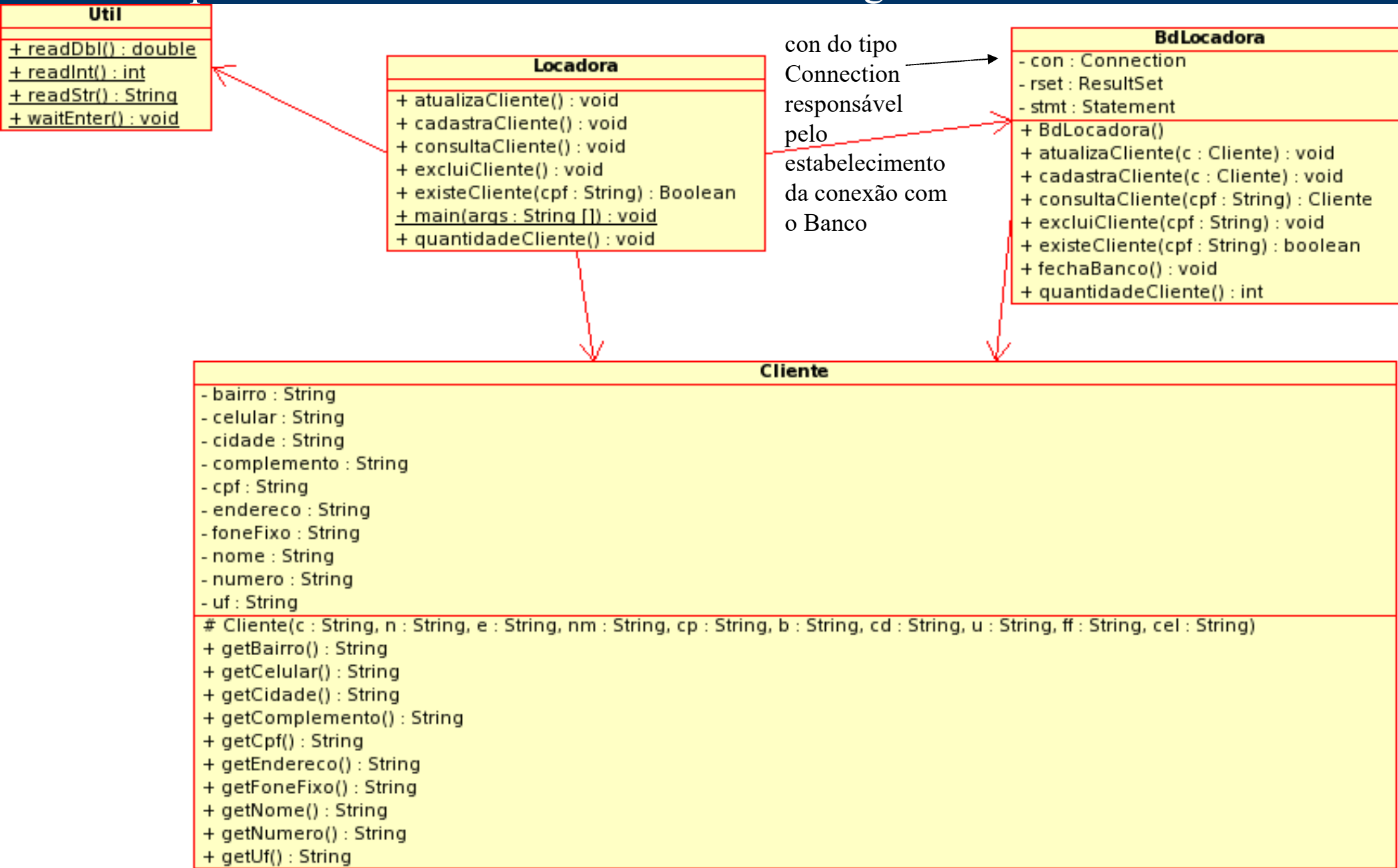
Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



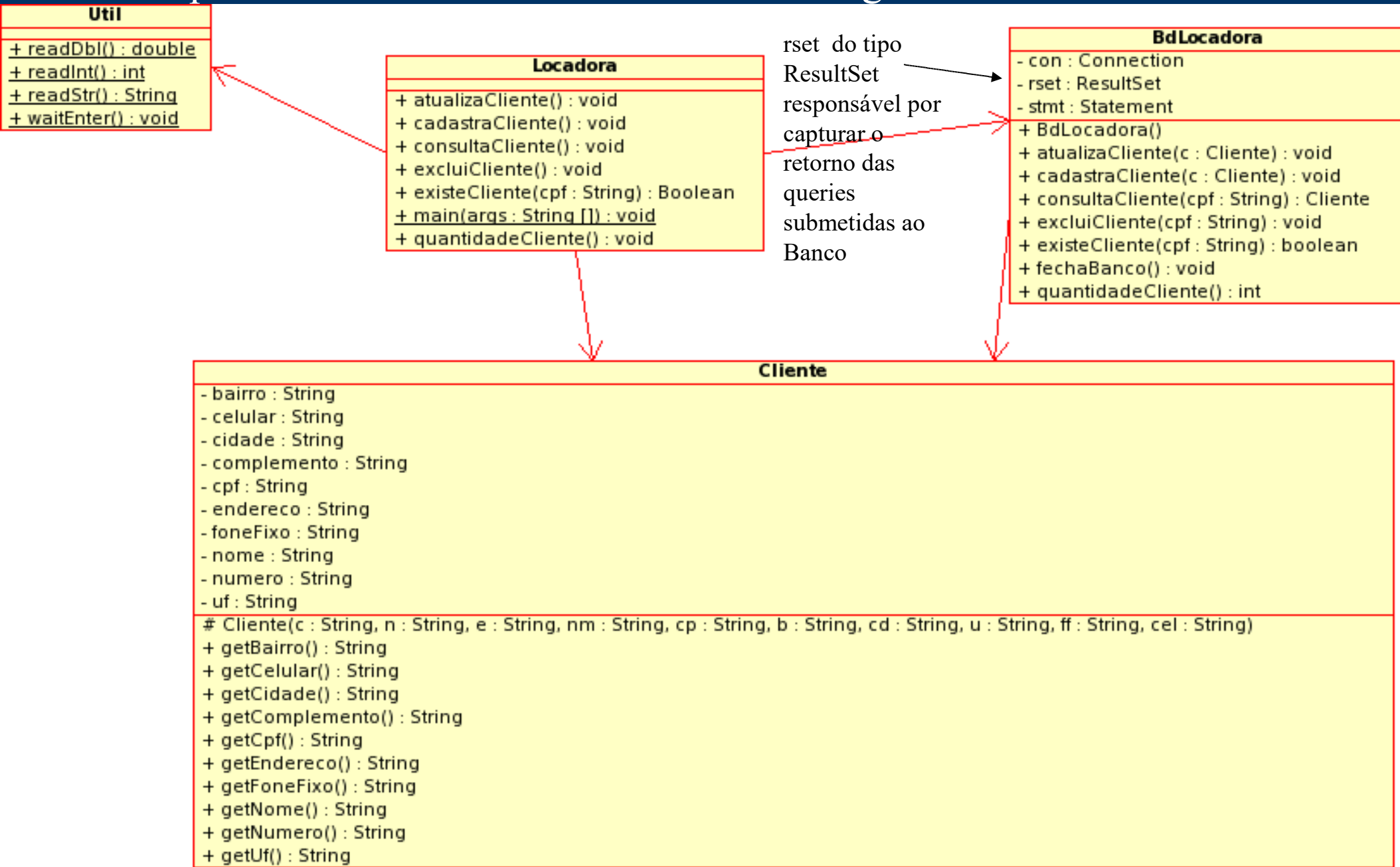
Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



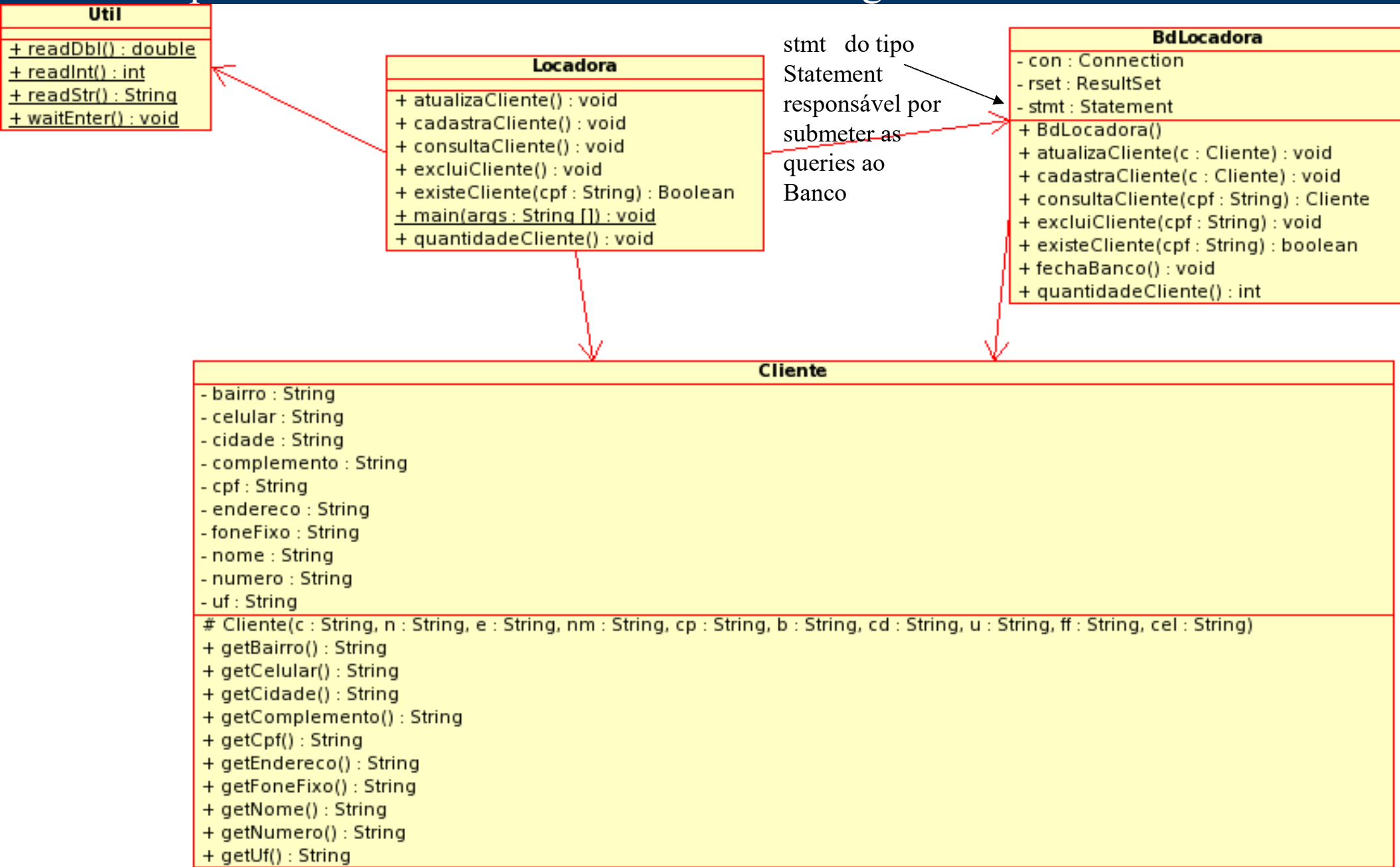
Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



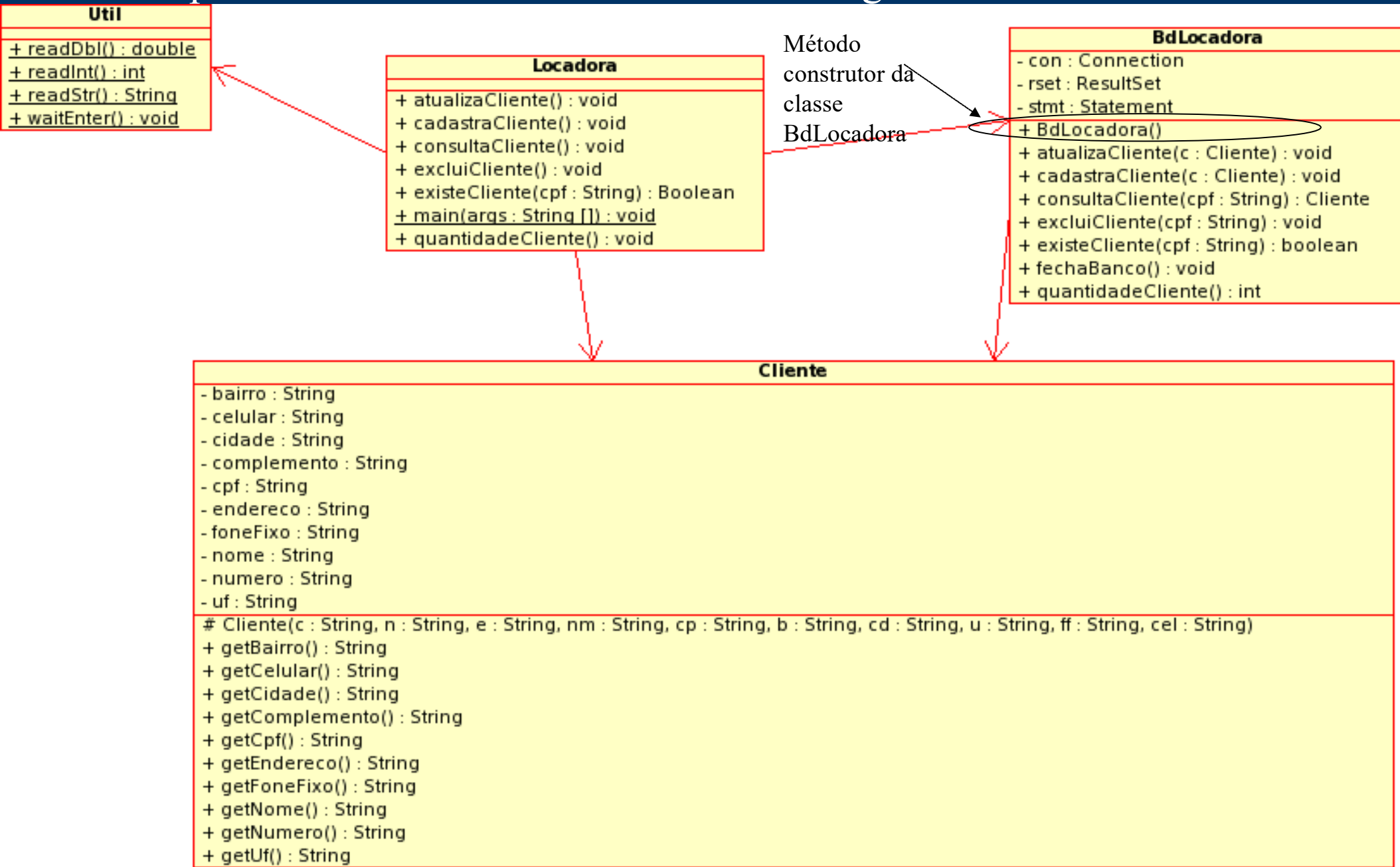
Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



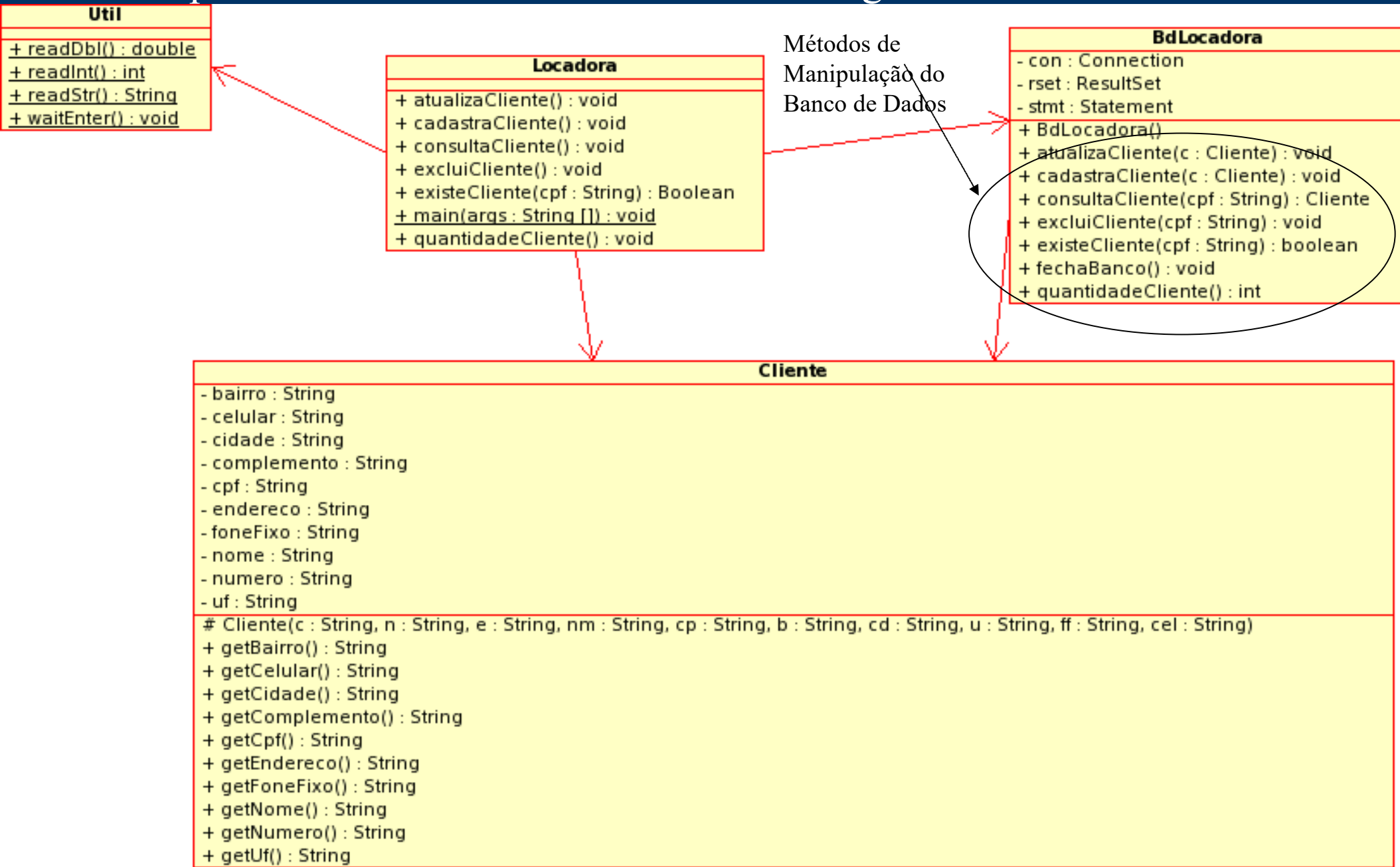
Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



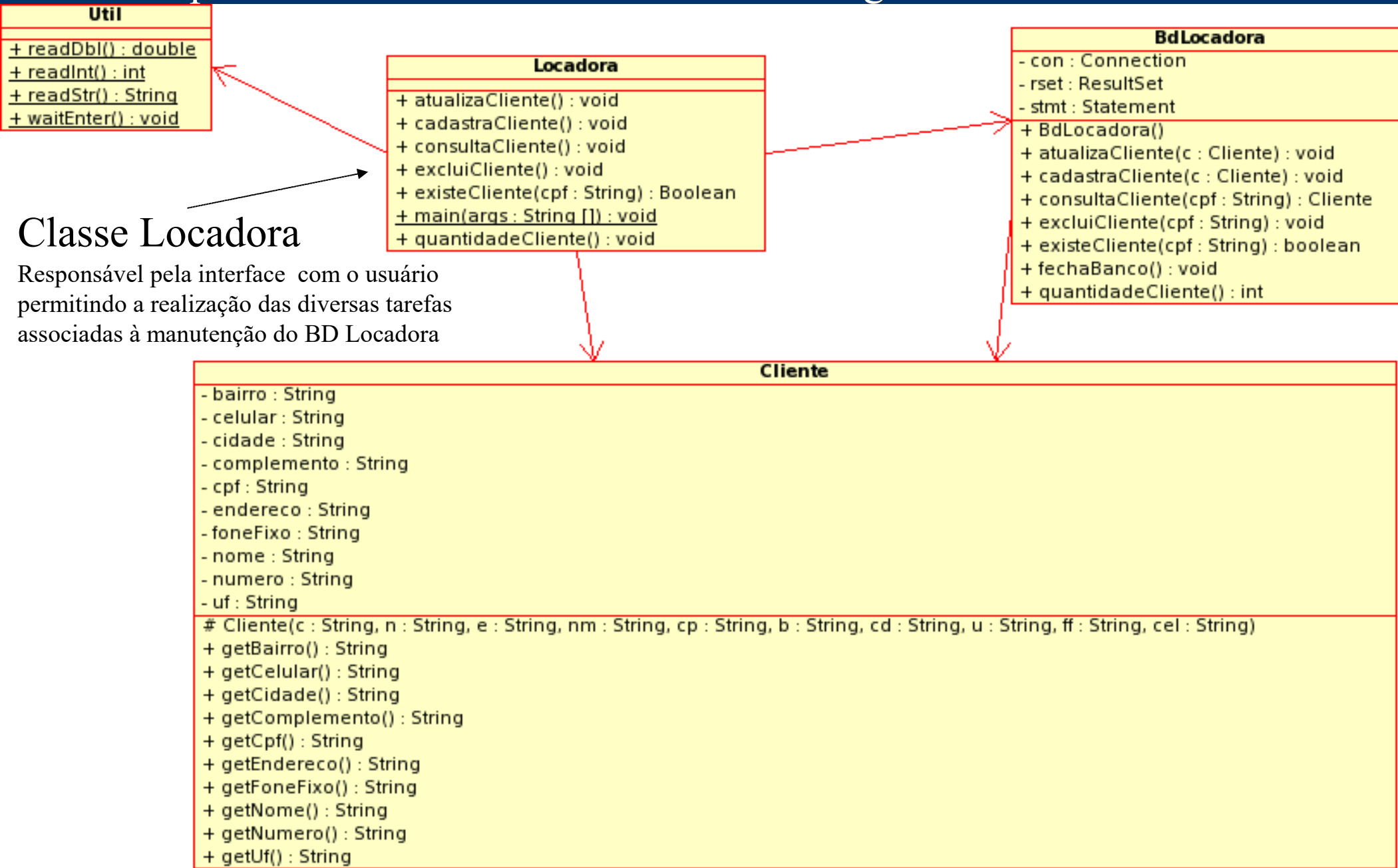
Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



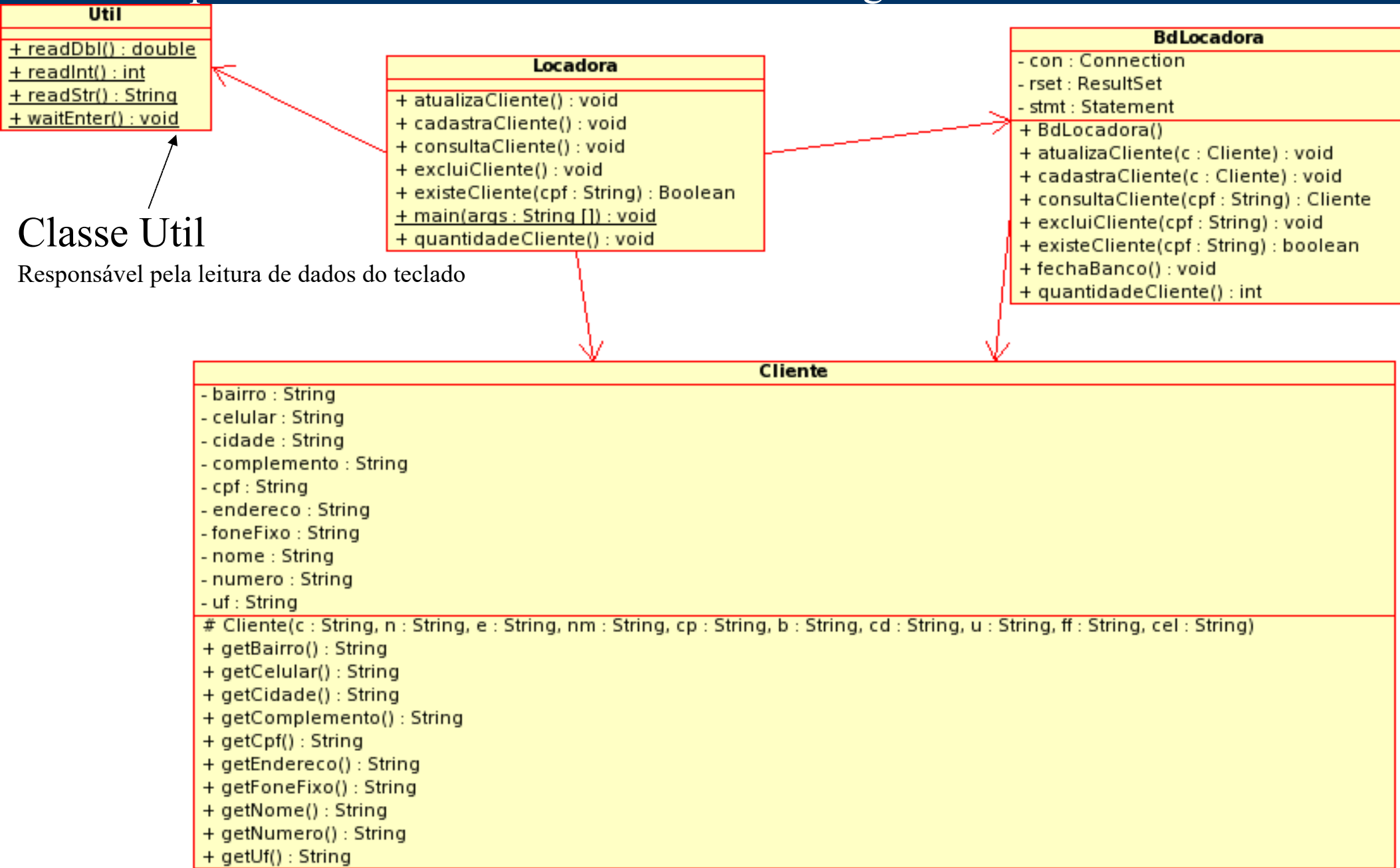
Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



Acesso a Banco de Dados

- Exemplo de Acesso a Banco em JAVA – Diagrama de Classes UML



Acesso a Banco de Dados

- Código-fonte disponível em

<http://www.uep.cnps.embrapa.br/~fabio/fape/PraticaOO/AcessoBancoDados/>

- Principais observações a respeito da realização de conexões ao Banco

- ✓ Abertura de conexão ao Banco

```
public BdLocadora() {  
    try {  
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
        String url = "jdbc:odbc:fape";  
        Connection con = DriverManager.getConnection(url,"raquel","");  
        stmt = con.createStatement();  
        System.out.println("Tentei criar conexão ODBC");  
    }  
    catch (SQLException e) {  
        System.out.println(e.getMessage());  
    }  
    catch (ClassNotFoundException e) {  
        System.out.println(e.getMessage());  
    }  
} // fim do construtor
```

Realizada dentro da estrutura try ... catch para tratar eventuais erros de conexão e tratá-los sem que o sistema aborte.

Acesso a Banco de Dados

- Código-fonte disponível em

<http://www.uep.cnps.embrapa.br/~fabio/fape/PraticaOO/AcessoBancoDados/>

- Principais observações a respeito da realização de conexões ao Banco
 - ✓ Inclusões de registros em tabelas do BD

```
public void cadastraCliente(Cliente c) {  
  
    try {  
  
        stmt.executeUpdate("INSERT INTO \"Cliente\" VALUES('" + c.getCpf() + "', '" + c.getNome() + "', '" + c.getEndereco() + "', '" + c.getNumero() + "', '" + c.getComplemento() + "', '" + c.getBairro() + "', '" + c.getCidade() + "', '" + c.getUf() + "', '" + c.getFoneFixo() + "', '" + c.getCelular() + "')");  
  
    }  
    catch (SQLException e) {System.out.println(e.getMessage());}  
  
} // fim do método
```

Acesso a Banco de Dados

- Código-fonte disponível em
<http://www.uep.cnps.embrapa.br/~fabio/fape/PraticaOO/AcessoBancoDados/>

- Principais observações a respeito da realização de conexões ao Banco
 - ✓ Exclusões de registros em tabelas do BD

```
public void excluiCliente(String cpf) {  
    try {  
        stmt.executeUpdate("Delete from \"Cliente\" Where cpf = '" + cpf + "'");  
    }  
    catch (SQLException e) {}  
  
}
```

Acesso a Banco de Dados

Código-fonte disponível em

<http://www.uep.cnps.embrapa.br/~fabio/fape/PraticaOO/AcessoBancoDados/>

- Principais observações a respeito da realização de conexões ao Banco
- ✓ Consulta registros em tabelas do BD

```
public Cliente consultaCliente(String cpf) {  
    try {  
        rset = stmt.executeQuery("Select * from \"Cliente\" where cpf = '" + cpf + "'");  
    }  
    catch (SQLException e) {System.out.println (e.getMessage());}  
    try {  
        if (rset.next()) { // inicialmente o ponteiro está posicionado antes do 1o. registro  
            String nome = rset.getString("nome");  
            String endereco = rset.getString("endereco");  
            String numero = rset.getString("numero");  
            String complemento = rset.getString("complemento");  
            String bairro = rset.getString("bairro");  
            String cidade = rset.getString("cidade");  
            String uf = rset.getString("uf");  
            String foneFixo = rset.getString("foneFixo");  
            String celular = rset.getString("celular");  
            Cliente c = new Cliente(cpf, nome, endereco, numero, complemento, bairro, cidade, uf, foneFixo, celular);  
            return c;  
        }  
    } //fim try  
    catch (SQLException e) {}  
    return null;  
} //fim do método
```

Acesso a Banco de Dados

Código-fonte disponível em
<http://www.uep.cnps.embrapa.br/~fabio/fape/PraticaOO/AcessoBancoDados/>

- Principais observações a respeito da realização de conexões ao Banco
 - ✓ Atualiza registros em tabelas do BD

```
public void atualizaCliente(Cliente c) {  
  
    try {  
        stmt.executeUpdate("Update \"Cliente\" set nome = '" + c.getNome() + "', " + "endereco = '" + c.getEndereco() + "', numero = '"  
+ c.getNumero() + "', complemento = '" + c.getComplemento() + "', bairro = '" + c.getBairro() + "', cidade = '" + c.getCidade() + "', uf  
= '" + c.getUf() + "', foneFixo = '" + c.getFoneFixo() + "', celular = '" + c.getCelular() + "' where cpf = '" + c.getCpf() + "'");  
    }  
    catch (SQLException e) {}  
}
```

Acesso a Banco de Dados

- Código-fonte disponível em
<http://www.uep.cnps.embrapa.br/~fabio/fape/PraticaOO/AcessoBancoDados/>

- Principais observações a respeito da realização de conexões ao Banco

- ✓ Fecha a conexão ao Banco

- ✓ Cada vez que uma conexão ao Banco é realizada, é importante fechar a conexão a fim de que não haja erros na próxima conexão a ser realizada

```
try {  
    ....  
    ....  
    stmt.close();  
    con.close();  
} catch {}
```

Acesso a Banco de Dados

- Código-fonte disponível em <http://www.uep.cnps.embrapa.br/~fabio/fape/PraticaOO/AcessoBancoDados/>
 - Principais observações a respeito do uso da Classe BdLocadora
 - ✓ Abstrai para o restante da implementação toda a complexidade que envolve a abertura e o fechamento de conexões ao Banco
 - ✓ A Classe Locadora torna-se “cliente” da classe BdLocadora quando precisa passar objetos a serem inseridos no Banco de Dados ou retirar objetos do Banco de dados
 - ✓ O Banco de Dados Relacional não insere objetos complexos no Banco. E.g. Um Cliente.
 - ✓ Insere ao invés itens de dados relacionados à entidade cliente identificada do mundo real
 - ✓ A Classe BdLocadora recupera itens de dados das entidades mantidas no Banco na forma de tabelas e “traduz” ou retorna objetos complexos para as demais classes da implementação em JAVA
-
-

Acesso a Banco de Dados

- Código-fonte disponível em <http://www.uep.cnps.embrapa.br/~fabio/fape/PraticaOO/AcessoBancoDados/>
- Principais observações a respeito do uso da Classe BdLocadora pela Classe Locadora

```
public void atualizaCliente() {  
    String cpf,nome,endereco,numero,complemento,bairro,cidade,uf,foneFixo,celular;  
  
    System.out.println("atualiza dados de Cliente >");  
    System.out.println("Informe o CPF do cliente a atualizar !");  
    System.out.print("CPF: ");  
    cpf = Util.readStr();  
    System.out.println("");  
  
    //if (existeCliente(cpf)) {  
  
    System.out.println("Nome");  
    nome = Util.readStr();  
    System.out.println("");  
  
    System.out.println("Endereço");  
    endereco = Util.readStr();  
    System.out.println("");
```

Dados do Cliente são lidos para posterior criação do objeto c da Classe Cliente que então é submetido ao método atualizaCliente do objeto bd da Classe BdLocadora

Acesso a Banco de Dados

- Código-fonte disponível em <http://www.uep.cnps.embrapa.br/~fabio/fape/PraticaOO/AcessoBancoDados/>
- Principais observações a respeito do uso da Classe BdLocadora pela Classe Locadora

```
System.out.println("Número");  
numero = Util.readStr();  
System.out.println("");
```

```
System.out.println("Complemento");  
complemento = Util.readStr();  
System.out.println("");
```

```
System.out.println("Bairro");  
bairro = Util.readStr();  
System.out.println("");
```

```
System.out.println("Cidade");  
cidade = Util.readStr();  
System.out.println("");
```

Dados do Cliente são lidos para posterior criação do objeto c da Classe Cliente que então é submetido ao método atualizaCliente do objeto bd da Classe BdLocadora

Acesso a Banco de Dados

- Código-fonte disponível em

<http://www.uep.cnps.embrapa.br/~fabio/fape/PraticaOO/AcessoBancoDados/>

Principais observações a respeito do uso da Classe BdLocadora pela Classe Locadora

```
System.out.println("Uf");  
uf = Util.readStr();  
System.out.println("");  
System.out.println("Fone Fixo");  
foneFixo = Util.readStr();  
System.out.println("");  
System.out.println("Celular");  
celular = Util.readStr();  
System.out.println("");
```

```
BdLocadora bd = new BdLocadora();  
Cliente c = new Cliente(cpf, nome, endereco, numero, complemento, bairro, cidade, uf, foneFixo, celular);  
bd.atualizaCliente(c);  
  
}
```

Dados do Cliente são lidos para posterior criação do objeto c da Classe Cliente que então é submetido ao método atualizaCliente do objeto bd da Classe BdLocadora

Referências

- JAVA 1001 Dicas de Programação. Mark C. Chan, Steven W. Griffith e Anthony F. Iasi. Makron Books 1999
 - ✓ Disponível na Biblioteca
 - ✓ Ótimo para dominar os conceitos de Orientação a Objetos
- JAVA Como Programar. J.M. Deitel e P.J. Deitel. Editora Bookman
 - ✓ Autor dos melhores livros de programação