

# Fundamentos da linguagem JAVA



## Aula 03

- ArrayList
- String
- StringBuffer

# Tópicos:



- **ArrayList**
- **Trabalhando com ArrayList**
- **Strings**
- **Operações com Strings**
- **StringBuffer**
- **Operações com StringBuffer**
- **Avaliação final da disciplina (aluno/professor)**

# ArrayList

O ArrayList é um gerenciador de coleções, ou seja, é um tipo de classe que guarda dados que podem ser acessados por índice.

Não é necessário estipular um tamanho para a lista.

Alta escalabilidade: aceita qualquer tipo de dado/objeto além de diferentes dados/objetos na mesma lista.

Por indexar objetos tem performance menor que o array.

# ArrayList



Contrutor:

```
ArrayList lista = new ArrayList();
```

Principais Métodos:

**add**([objeto/dado]) : adiciona um item na lista

**clear**( ) : limpa toda a lista

**get**([int]): retorna o objeto/dado na posição informada pelo índice.

Se o índice não existir retorna exceção.

Sempre será obrigatório a conversão para o objeto/dado ao qual se está acessando.

**isEmpty**() : retorna true se a lista estiver vazia

**remove**([int]): Retira um determinado item da lista.

**Size**( ) : Retorna o tamanho (qtade de itens) do ArrayList.

# ArrayList

Exemplo:

```
ArrayList lista = new ArrayList();
int op=1;
String retorno="";
while(op==1){
    Pessoa p = new Pessoa();
    p.setNome(JOptionPane.showInputDialog("Nome: "));
    lista.add(p);
    op=Integer.parseInt(JOptionPane.showInputDialog("Continuar 1 - Sim"));
}
for(int i=0;i<lista.size();i++){
    Pessoa p = (Pessoa)lista.get(i);
    retorno+=p.getNome()+"\n";
}
JOptionPane.showMessageDialog(null,retorno);
System.exit(0);
```



# ArrayList

## Exercício:

Elaborar um programa que armazene em um ArrayList informações relativas à produtos (o usuário decidirá quando deve parar de inserir produtos)

Nome - Valor – Rotatividade (1 – alto giro / 2 – baixo giro)

Após o carregamento das informações, retornar:

Qual produto de baixo giro de valor mais alto

Qual produto de alto giro de valor mais baixo

O somatório dos valores (alto giro, baixo giro e total);

**ATENÇÃO:** (classes necessárias)

- Uma classe para produto.

- Uma classe para leitura e retorno

- Uma classe **ESTÁTICA** para processamento (passar o ArrayList como parâmetro).

# Strings

Um String é uma sequência de caracteres (instância de objeto da classe String).

Possui métodos mais “seguros” do que array de caracteres;

Objetos da classe String são do tipo final, ou seja não podem ser alterados depois de criados.

Quando alteramos o conteúdo de uma String, na verdade, outro objeto é criado.

# Strings

Por exemplo:



```
String str = "Poxa vida! ";  
str += "Que coisa estranha";
```

Na 1ª linha o objeto String é criado com o valor "Poxa vida!" e a variável str referencia este objeto.

Na 2ª linha, é criado um OUTRO objeto, contendo o valor " Poxa vida! Que coisa estranha" e a variável 'str' é reatribuída de modo a referenciar a nova String. Note que a String que continha somente o "Poxa vida!" ainda existe, mas não é referenciada pela variável 'str' (tornando-se elegível para coleta de lixo).

Ou seja, o objeto String não têm o seu valor alterado: sempre que se mexe numa string, um novo objeto é criado.



# Strings

Por exemplo:

```
public static void main(..) {  
    String str = "Teste1";  
    teste(str);  
    System.out.println(str); // Imprime "Teste1" !!!  
}  
public static void teste(String s) {  
    s+= ", Teste2";  
}
```

Quando se passa uma String por parâmetro, a variável 's' ainda referencia o mesmo objeto referenciado por 'str', criado no início do método main.

Quando se modifica a String dentro do método 'teste', um novo objeto é criado dentro do escopo do método, com o valor "Teste1, Teste2". Então, quando o método termina de ser executado, este objeto deixa de existir. A string original não foi modificada, e ainda é referenciada pelo objeto 'str'.

# Principais métodos Strings



O método `length()` retorna o tamanho da string.

```
System.out.println("Hello".length()); // imprime 5
```

O operador “+” é utilizado para concatenar 2 ou mais strings.

```
String meunome = "Harry"  
String str = "Meu nome é" + meunome + ".";
```

Para concatenação de strings o compilador Java converte um operando para uma String sempre que o outro operador da operação “+” for um objeto String.

# Principais métodos Strings



`public char charAt(int índice):` Retorna o caracter na posição dada pelo índice.

`public int compareTo(String s2):` Compara léxico-graficamente duas Strings, Retorna 0 se ambas forem iguais, um número negativo se a String 1 for menor que a String 2 e um número positivo ao inverso.

Ex: `"a".compareTo("b");` // retorna -1

`public int compareToIgnoreCase(String s2):` Idem ao anterior ignorando maiúsculas e minúsculas.

`public String concat(String s2):` Concatena String1 com a String2

`public static String copyValueOf(char[] c):` retorna uma String com o conteúdo do array de caracteres c.

# Principais métodos Strings



`public boolean equals(Object o)`: Verifica se o objeto (o) é uma String com a mesma sequência de caracteres.

`public boolean equalsIgnoreCase(String s2)`: Compara com a string s2 ignorando as diferenças entre maiúsculas e minúsculas.

`Public int IndexOf(...)` – Parâmetros aceitos:

`int c`: Retorna o índice da primeira ocorrência do caractere c.

`int c, int inicio`: Retorna o índice da primeira ocorrência do caractere c a partir da posição inicio.

`String str`: Retorna o índice da primeira ocorrência da sub-string str.

`String str, int inicio`: Retorna o índice da primeira ocorrência da sub-string str a partir da posição inicio.

# Principais métodos Strings



`public String replace(char c1,char c2 )` – retorna uma nova String substituindo todas as ocorrências do caractere c1 pelo caractere c2.

`public String substring(int inicio)` – Retorna uma substring (como uma nova String) incluindo os caracteres à partir da posição início (inclusive).

Ex: String str: “complicação”;

`str = str.substring(7); // ação`

`public String substring(int inicio, int fim)` – Retorna uma substring (como uma nova String) incluindo os caracteres à partir da posição início (inclusive) e terminando no caractere anterior à posição fim.

Ex: String str: “anatomicamente”

`str = str.substring(2,9); // atomica`

# Principais métodos Strings



`public char[] toCharArray( )` – retorna um array de caracteres com os caracteres da String;

`public String toLowerCase( )` - Converte para minúsculas.

`public String toUpperCase( )` - Converte para maiúsculas.

`public String trim( )` – Elimina os espaços em branco do início e do fim.

`public boolean startsWith(String s1)` – retorna true se uma String inicia com um prefixo especificado;

`public boolean endsWith(String s1)` – retorna true se uma String termina com um prefixo especificado;

# Principais métodos Strings



public static String **valueOf(...)** – Parâmetros aceitos:

boolean

char[]

char

int

long

double

float

Object

Retorna o mesmo que o `.toString()`. //conversão para String;

# Exercício 1 Strings

Criar um programa que leia duas senhas informadas pelo usuário:

Senha

Confirmação de senha

Faça os testes e valide ou não as senhas:

- .Devem ser iguais
- .Devem possuir no mínimo 8 caracteres
- .O terceiro caractere não pode ser a letra C (maiúscula)
- .O terceiro caractere não pode ser a letra a (minúscula)
- .É obrigatório a presença de do mínimo um caractere Y (indiferente de maiúsculo ou minúsculo)



# Exercício 2 Strings

Criar um programa que leia uma entrada em formato unicode (utilizado em arquivos TXT) com a seguinte formatação:

00000000000000000000

4 caracteres: código da mercadoria

4 caracteres: quantidade pedida

4 caracteres: percentual de desconto (0935 = 9.35%)

6 caracteres: Valor do produto (001250=12.50)

Mostrar

Código

Quantidade

Total bruto (sem desconto)

Total líquido (com desconto)

# StringBuffer

A classe StringBuffer representa uma sequência de caracteres não constante (diferente da String que é constante).

Sendo assim é possível a alteração de caracteres de objetos desta classe sem a realocação de outro objeto.

Muitas vezes ao lidar com concatenações de Strings o compilador transforma temporariamente algumas delas em StringBuffer(s).

A sun indica a utilização de StringBuffer ao invés de String quando for necessário trabalhar com Strings não-constantes, evitando assim, realocação constantes de objetos.

# StringBuffer

A inicialização direta por atribuição, como em String, não é possível.

Concatenação com o operador + não é possível. Deve ser feita com os métodos específicos.

```
//String
```

```
String s = "Meu nome é:";
```

```
s += " João"; // s assume o valor "Meu nome é: João"
```

```
//StringBuffer
```

```
StringBuffer sb1 = new StringBuffer("Meu nome é:");
```

```
sb1.append(" João"); // sb1 assume o valor "Meu nome é: João"
```

```
StringBuffer sb2 = new StringBuffer(" João");
```

```
sb2.insert(0,"Meu nome é:"); // sb2 assume o valor "Meu nome é: João"
```

# StringBuffer

## Contrutores:

`new StringBuffer()` – Constrói um StringBuffer com capacidade inicial para armazenar 16 caracteres.

`new StringBuffer(int tamanho)` – Constrói um StringBuffer com capacidade inicial para armazenar o tamanho definido.

`new StringBuffer(String s)` – Constrói um StringBuffer à partir de uma String.

# StringBuffer

## Principais métodos:

public StringBuffer **append**(...) – Append sempre acrescenta ao final do objeto corrente. Parâmetros aceitos:

boolean b

char c

char[] c

double d

float f

int i

long l

Object o (o.toString)

String s

# StringBuffer

## Principais métodos:

public int **capacity**( ) – retorna a capacidade corrente do StringBuffer. Se o armazenamento exigir maior capacidade, uma realocação irá ocorrer.

public char **charAt**(int índice) – retorna um caracter na posição definida pelo índice.

public StringBuffer **delete**(int inicio, int fim) – Remove do StringBuffer os caracteres entre as posições início(inclusive) e fim(inclusive). Retorna uma referencia ao objeto corrente.

public StringBuffer **deleteCharAt**(int índice) – remove um carater específico na posição índice.

# StringBuffer

## Principais métodos:

public StringBuffer **insert**(int início, ...) – insere no objeto corrente, à partir da posição início, uma representação (...) dos mesmos parâmetros aceitos pelo append.

public int **length**( ) – Retorna o número de caracteres do StringBuffer.

public StringBuffer **replace**( int início, int fim, String s) – substitui os caracteres entre as posições início (inclusive) e fim (inclusive) pela String s.

public StringBuffer **reverse**() – Inverte a posição de todos os caracteres.

public void **setCharAt**(int índice, char c) – substitui o caracter da posição índice pelo caractere c.

# StringBuffer

## Principais métodos:

public String **substring**(int início) – retorna uma String contendo os caracteres à partir da posição início(inclusive)

public String **substring**(int início,int fim) – retorna uma String contendo os caracteres entre as posições início(inclusive) e antes da posição fim.



# StringBuffer

## Exemplo

```
StringBuffer sb = new StringBuffer("Hello");  
sb.length(); // 5  
sb.capacity(); // 21 (16 caracteres + 5 da String)  
sb.charAt(1); // e  
sb.setCharAt(1, 'i'); // Hillo  
sb.setLength(2); // Hi  
sb.append("l").append("l"); // Hill  
sb.insert(0, "Big "); // Big Hill
```

# AVALIAÇÕES

