Fundamentos de Java

Strings, Datas e Números

SoftBlue
www.softblue.com.br

Todos os direitos de cópia reservados. Não é permitida a distribuição física ou eletrônica deste material sem a permissão expressa e por escrito do autor.

Tópicos Abordados

- Strings
  - Strings na memória
  - Métodos importantes da classes *String*
- *StringBuilder*
- Formatando strings
- Trabalhando com datas
  - Formatação de data
- Formatando números
  - Formatação de moeda
- Números randômicos

Strings

- Armazenam conjuntos de caracteres
- As strings são objetos, logo podem ser construídas como qualquer outro objeto

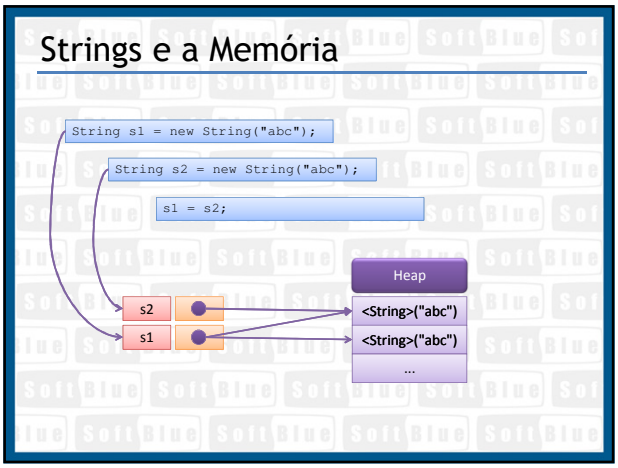
```
String s = new String();
```
String vazia

```
String s = new String("abc");
```
String "abc"

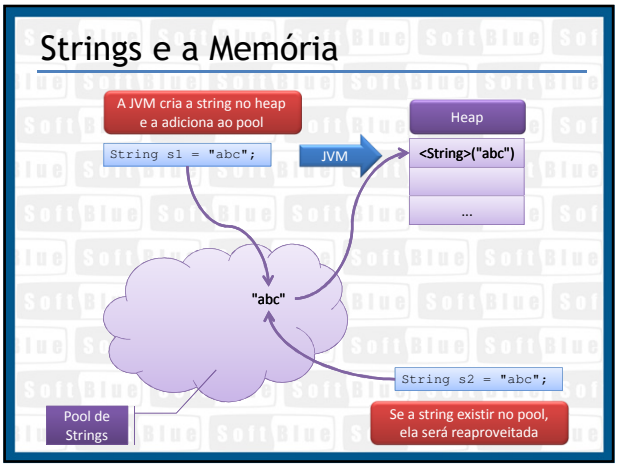
```
String s = "abc";
```
String "abc"

Qual a diferença?

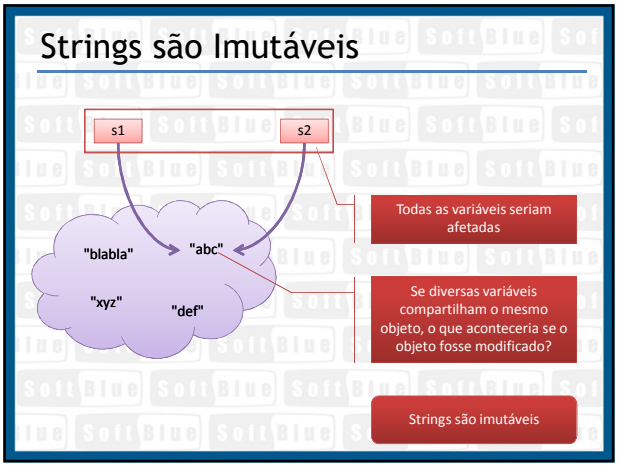
Strings e a Memória



Strings e a Memória



Strings são Imutáveis



Strings são Imutáveis

- Depois de criada, uma string nunca tem seu valor alterado

```
String s = "abc";
s.toUpperCase();
```
s continua com o valor "abc"

```
String s = "abc";
s = s.toUpperCase();
```
s deixa de ser "abc" e referencia uma nova string, "ABC"

```
String s = "abc";
s.concat("def");
```
s continua com o valor "abc"

```
String s = "abc";
s = s.concat("def");
```
s deixa de ser "abc" e referencia uma nova string, "abcdef"

Trabalhando com Strings

- O operador "+" pode ser utilizado na concatenação de strings

```
String s1 = "abc";
String s2 = "123" + s1;
```
s2 passa a ter o valor "123abc"

- Para comparar strings, o método **equals()** deve ser utilizado

```
if (s1.equals(s2)) {
    ...
}
```
O *equals()* compara o conteúdo ao invés de comparar endereços de memória

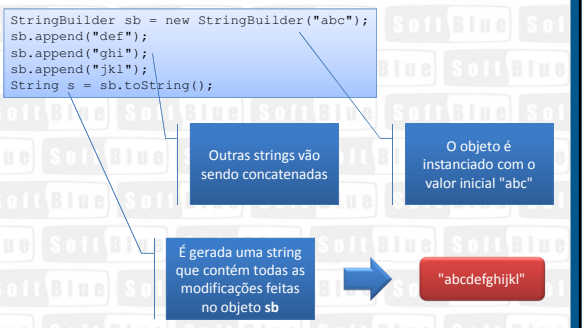
Métodos da Classe String

Método	Descrição
charAt(int)	Retorna o caractere de uma posição
indexOf(String)	Retorna a posição em que uma string aparece pela primeira vez na string principal
length()	Retorna o tamanho da string
split(String)	Divide a string de acordo com um critério
substring(int, int)	Retorna uma parte da string
toLowerCase()	Converte os caracteres para minúsculo
toUpperCase()	Converte os caracteres para maiúsculo

StringBuilder

- Como strings são imutáveis, manipular a mesma string diversas vezes pode ocupar muita memória desnecessariamente
  - Bastante comum em concatenação de strings dentro de um loop
- A classe **StringBuilder** resolve este problema
  - Existe também a classe **StringBuffer**, que tem exatamente a mesma função mas que possui todos os seus métodos sincronizados

Usando a Classe StringBuilder



Métodos da Classe StringBuilder

Método	Descrição
append(String)	Concatena uma string
delete(int, int)	Remove parte de uma string
insert(int, String)	Insere uma string em uma determinada posição
reverse()	Inverte os caracteres
toString()	Retorna o conteúdo do objeto como uma string

Formatando Strings

- A formatação de strings pode ser feita facilmente através dos métodos **format()** e **printf()** da classe **PrintWriter**
  - Ambos os métodos funcionam da mesma forma
  - *System.out* é um *PrintWriter*, portanto é possível formatar a saída para o console

Sintaxe → printf("<string>", <argumentos>)

Formatando Strings

```
System.out.printf("%d, %f", 245, 100.0);
```
→ `245, 100,000000`

```
System.out.printf("%.2f", 100.0);
```
→ `100,00`

```
System.out.printf(">%7d<\n>%7s<", 2000, "abc");
```
→
```
>   2000<
>    abc<
```

```
System.out.printf("%05d", 25);
```
→ `00025`

Trabalhando com Datas

- Java possui quatro classes principais para trabalhar com datas

Classe	Descrição
java.util.Date	Representa uma data e hora
java.util.Calendar	Possibilita a conversão e manipulação de datas e horas
java.text.DateFormat	Formata datas e horas
java.util.Locale	Representa uma localidade. É utilizada com datas para formatá-las de acordo com a localidade desejada.

Exemplos no Uso de Datas

- Obter a data/hora atual

```
Date d = new Date();
System.out.println(d.toString());
```

```
Thu Oct 29 18:58:02 BRST 2009
```

- Somar 7 dias à data atual

```
Calendar c = Calendar.getInstance();
c.add(Calendar.DAY_OF_MONTH, 7);
Date d = c.getTime();
System.out.println(d.toString());
```

```
Thu Nov 05 18:58:02 BRST 2009
```

Exemplos de Formatação de Datas

- Formatação da data atual no padrão curto

```
Date d = new Date();
DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT);
String s = df.format(d);
System.out.println(s);
```

```
29/10/2009
```

Exemplos de Formatação de Datas

- Formatação da data atual no padrão longo, de acordo com o francês falado na França

```
Date d = new Date();
Locale l = new Locale("fr", "FR");
DateFormat df = DateFormat.getDateInstance(DateFormat.LONG, l);
String s = df.format(d);
System.out.println(s);
```

```
29 octobre 2009
```

Para formatar a hora, use o **getTimeInstance()**

Formatando Números

- Java possui a classe **NumberFormat**, utilizada para formatar números
- Esta classe funciona de forma semelhante à *DateFormat*
- Também tem suporte à localização

Exemplo de Formatação Numérica

- Formatação do número, considerando separadores de milhar e casas decimais

```
NumberFormat nf = NumberFormat.getInstance();
String s = nf.format(1000.5);
System.out.println(s);
```

```
1.000,5
```

- Agora no padrão americano

```
Locale l = new Locale("en", "US");
NumberFormat nf = NumberFormat.getInstance(l);
String s = nf.format(1000.5);
System.out.println(s);
```

```
1,000.5
```

Exemplos de Formatação de Moeda

- Formatação de moeda no padrão brasileiro

```
Locale l = new Locale("pt", "BR");
NumberFormat nf = NumberFormat.getCurrencyInstance(l);
String s = nf.format(1000.5);
System.out.println(s);
```

```
R$ 1.000,50
```

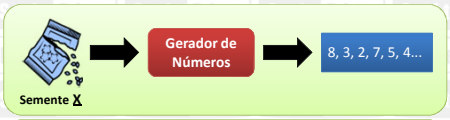
- Agora no padrão italiano

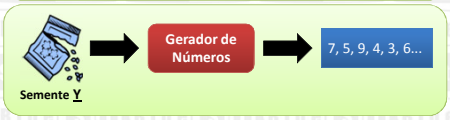
```
Locale l = new Locale("it", "IT");
NumberFormat nf = NumberFormat.getCurrencyInstance(l);
String s = nf.format(1000.5);
System.out.println(s);
```

```
€ 1.000,50
```

Números Randômicos

- Java é capaz de gerar números randômicos, que na verdade são pseudo-randômicos





Sementes iguais geram sequências iguais

A Classe *Random*

- A classe *Random* pode ser utilizada para gerar números randômicos

```
Random r = new Random();
```

Cria um gerador de números randômicos. É possível especificar uma semente

```
double n = r.nextDouble();
```

Gera um novo número *double*. É possível gerar números de outros tipos

```
int n = r.nextInt(10);
```

Gera um novo *int* entre 0 e 9

O Método *Math.random()*

- Outra opção é utilizar o método *Math.random()*
  - Gera números do tipo *double*
  - Os números são distribuídos entre 0 e 0,99999...

```
double n = Math.random();
```

Gera o próximo *double* da sequência. Utiliza internamente a classe *Random*

O Método *Math.random()*

- Como implementar um método que define um intervalo de geração de números?

```
int gerarInt(int inicio, int fim) {
  int intervalo = fim - inicio;
  int n = (int)(Math.random() * intervalo) + inicio;
  return n;
}
```

Algoritmo

1. Calcula o intervalo entre os números
2. Gera um *double* randômico entre 0 e 0,9999...
3. Multiplica o valor pelo intervalo
4. Trunca as casas decimais
5. Soma o resultado com o início do intervalo

Colocando em Prática...

Agora que você já aprendeu a teoria, acesse as vídeo-aulas práticas e pratique os assuntos abordados neste módulo!

Clique aqui para acessar as vídeo-aulas práticas