Programação Orientada a Objetos

Professor Filipe Dwan Pereira

Aula 10 – Strings, Números e Datas

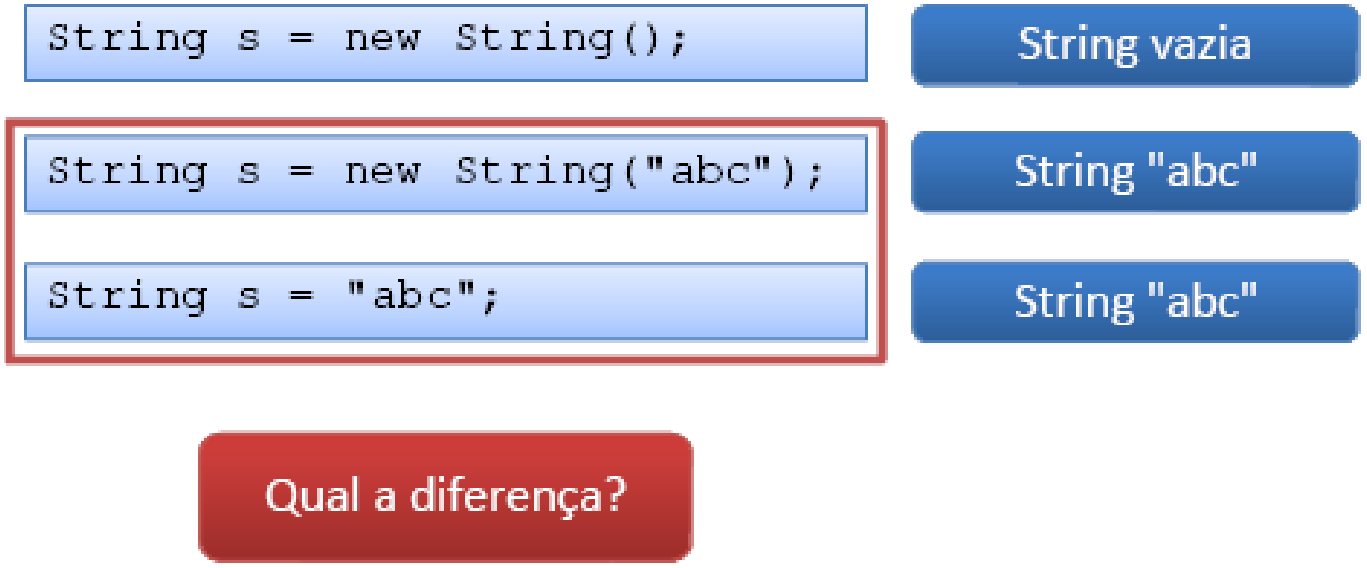
> *"...o homem não pode receber coisa alguma, se lhe não for dada do céu".*
>
> *João 3:27*

Tópicos Abordados

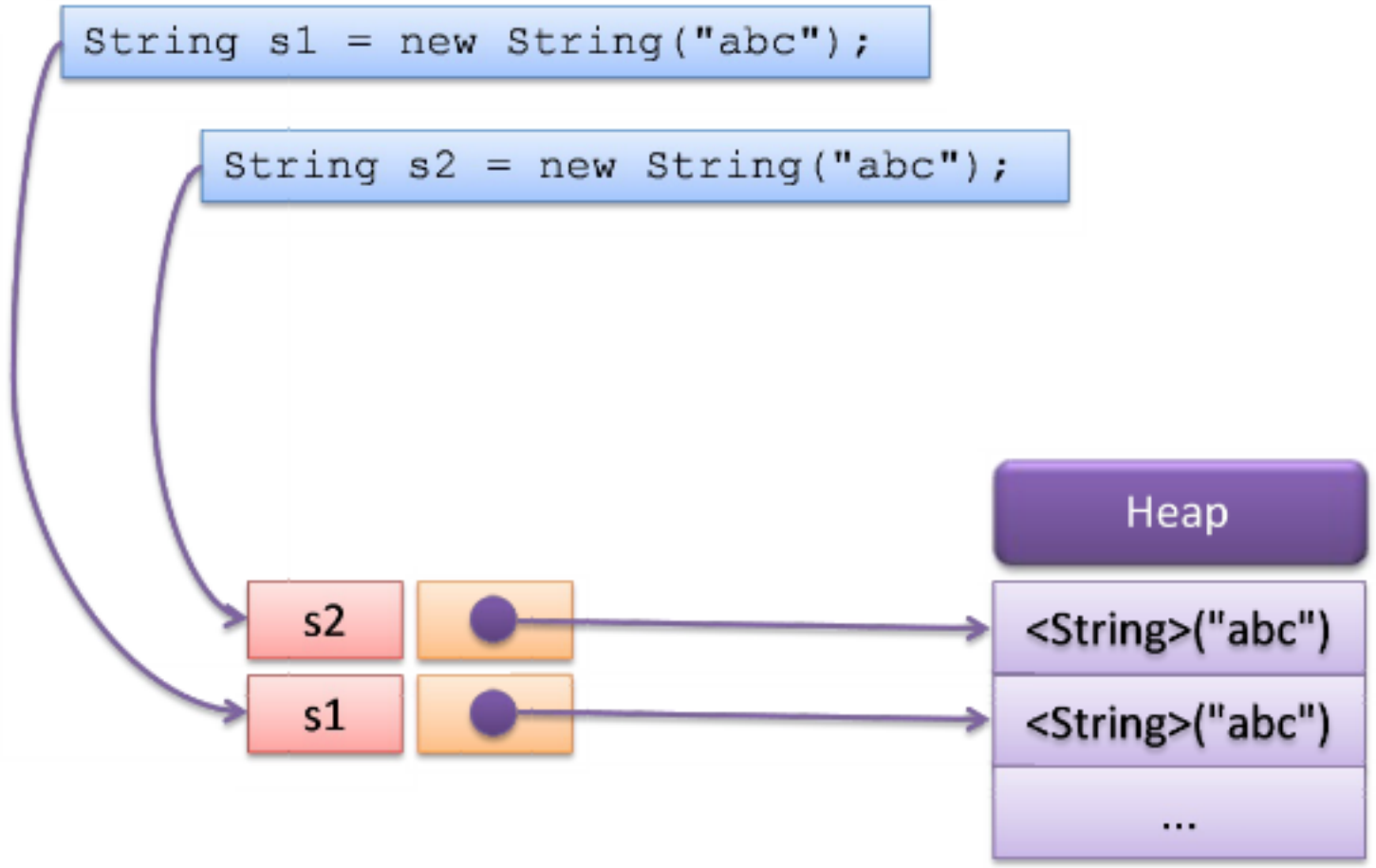
- Strings
  - Strings na memória
  - Métodos importantes da classes String
- StringBuilder
- Formatando strings
- Trabalhando com datas
  - Manipulação de datas
  - Date and Time API (pacote java.time)
- Formatando números
  - Formatação de moeda
- Números randômicos
- Métodos importantes da classe Math

Strings

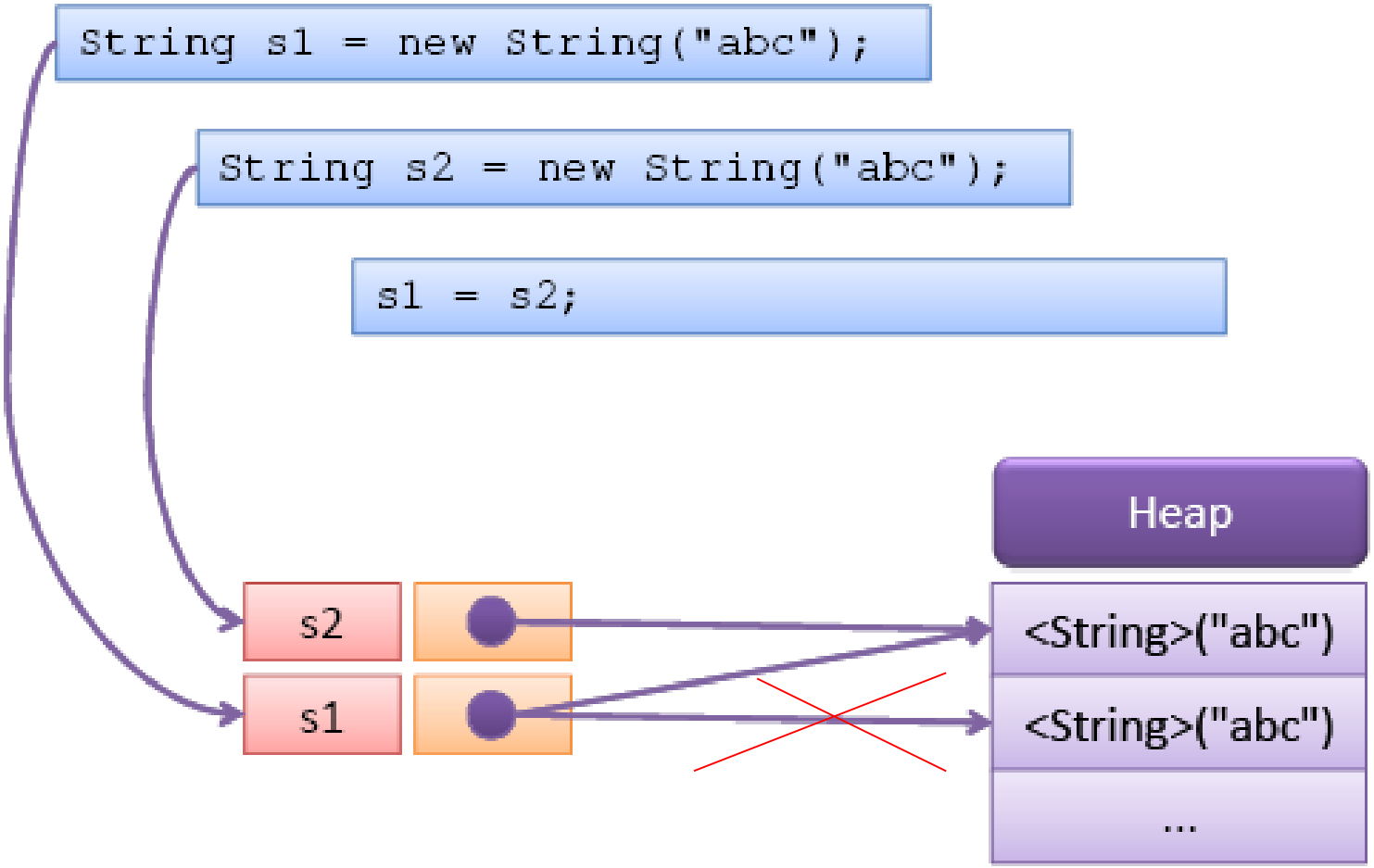
- Armazenam conjuntos de caracteres
- As strings são objetos, logo podem ser construídas como qualquer outro objeto



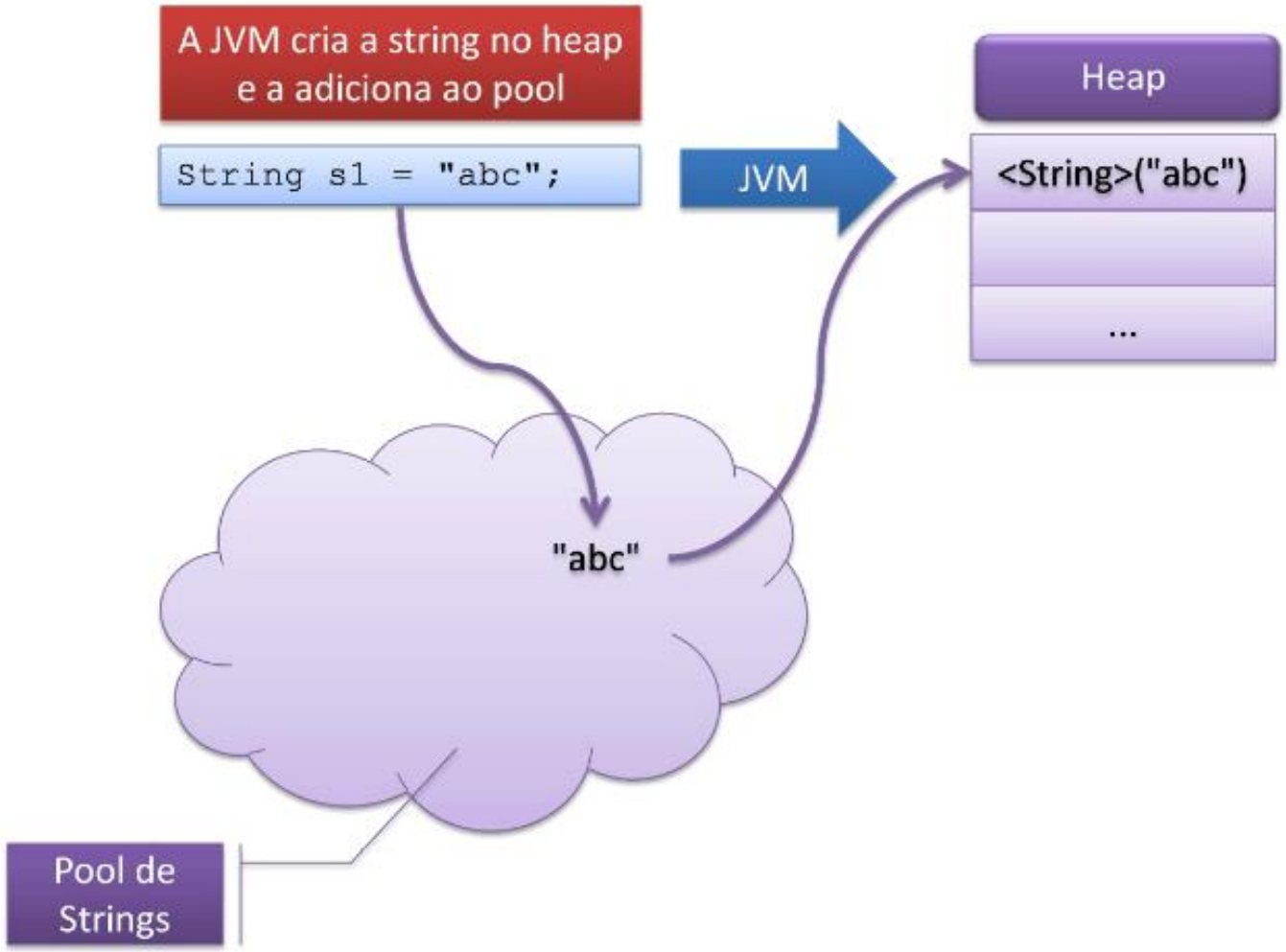
Strings e a Memória



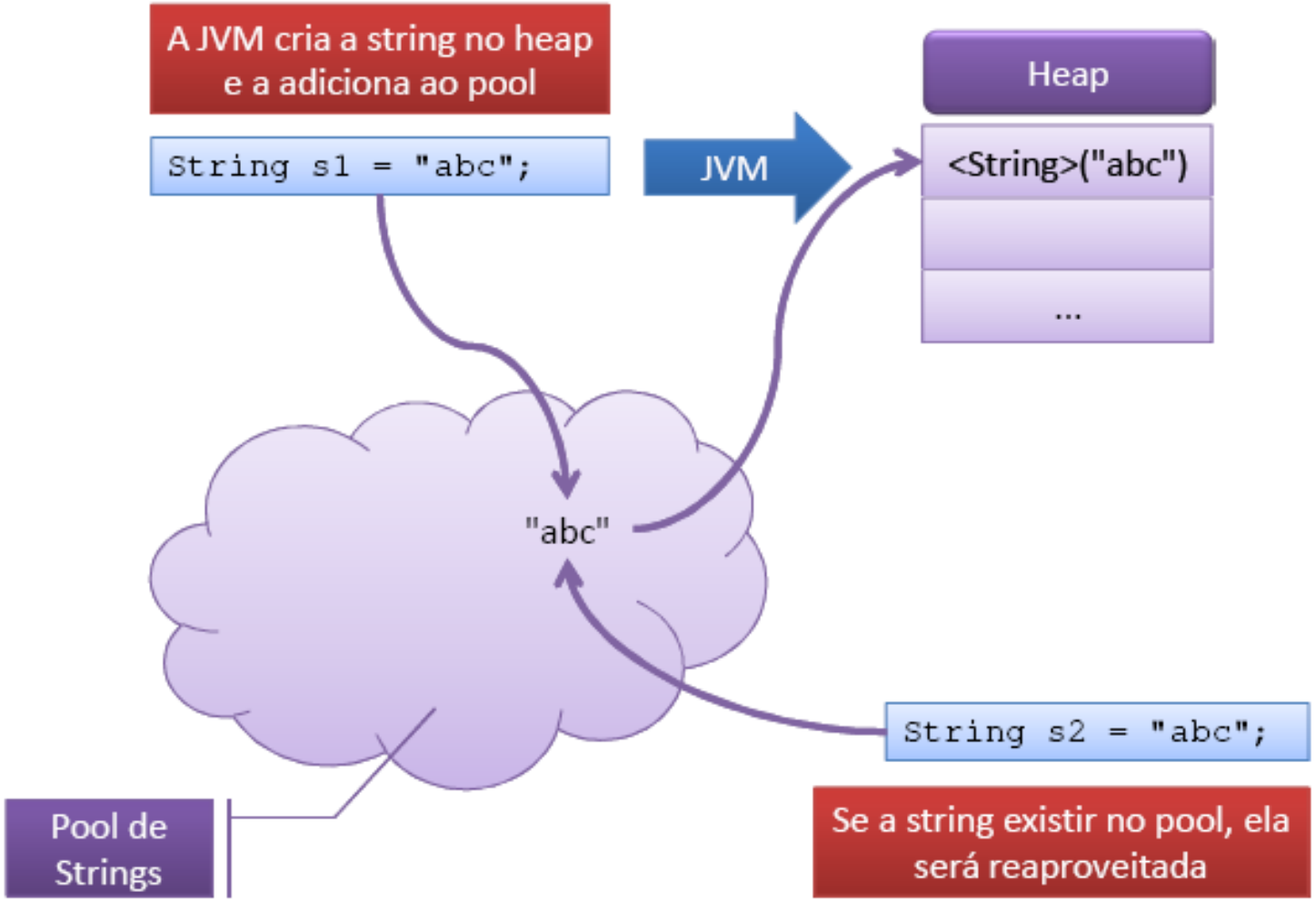
Strings e a Memória



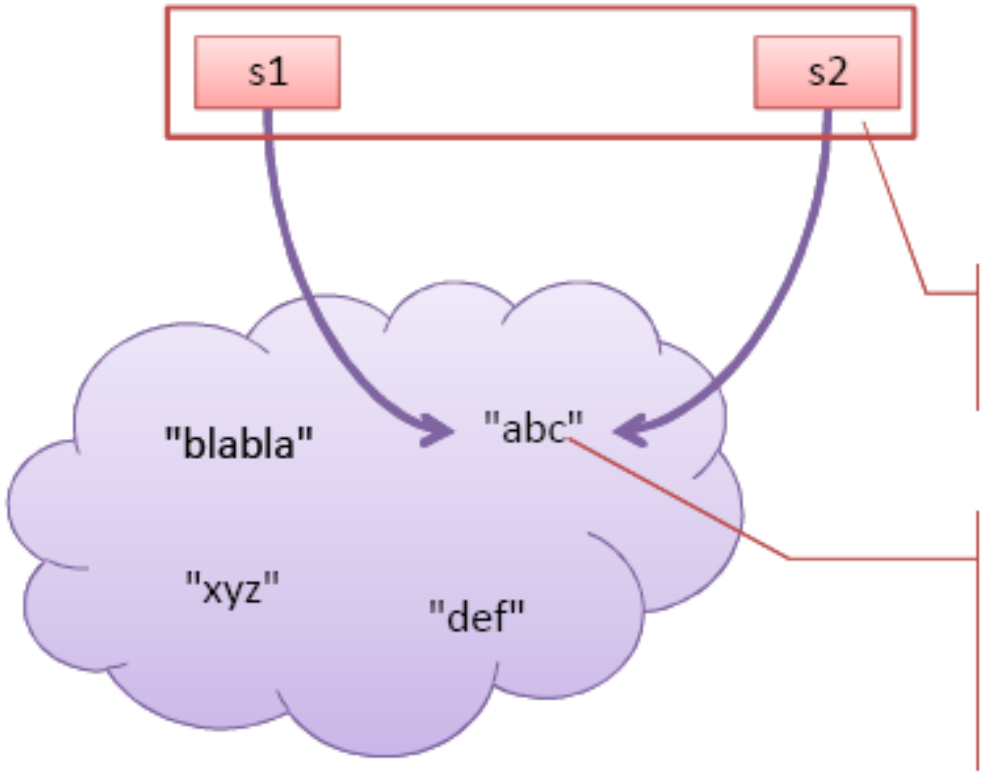
Strings e a Memória



Strings e a Memória



Strings são Imutáveis



Todas as variáveis seriam afetadas

Se diversas variáveis compartilham o mesmo objeto, o que aconteceria se o objeto fosse modificado?

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

Exemplo1:

```
public class TesteStrings {

    public static void main(String[] args) {
        String s1 = new String("abc");
        String s2 = "abc";
        String s3 = "abc";

        System.out.println(s1==s2);
        System.out.println(s2==s3);

        System.out.println(s1.equals(s2));
        System.out.println(s2.equals(s3));

        s2.toUpperCase();
        System.out.println(s2);

        s2 = s2.toUpperCase();
        System.out.println(s2);

        s2 = s2.concat(s3);
        s2 = s2+s3;
        System.out.println(s2);
    }

}
```

Exemplo2:

```
public class UsandoString1 {
  public static void main(String[] args) {
      String s =  " Voar voar, sorrir sorrir ";
      System.out.println("O tamanho da String é: " + s.length());
      System.out.println("O caracter da posição 7 é: " + s.charAt(7));
      System.out.println("String em caixa alta: " + s.toUpperCase());
      System.out.println("String em caixa baixa: " + s.toLowerCase());
      System.out.println("A substring de 2 a 8 é: " + s.substring(2, 8));
      System.out.println("Tirando os espaços em branco antes e depois temos:" + s.trim());
      String frase1 = s.replace("sorrir", "latir");
      System.out.println(frase1);
      int num = 10;
      frase1 = String.valueOf(num);
      System.out.println(frase1);
      System.out.println(s.indexOf("sorrir"));
      System.out.println(s.indexOf("sorrir", 15));
  }
}
```

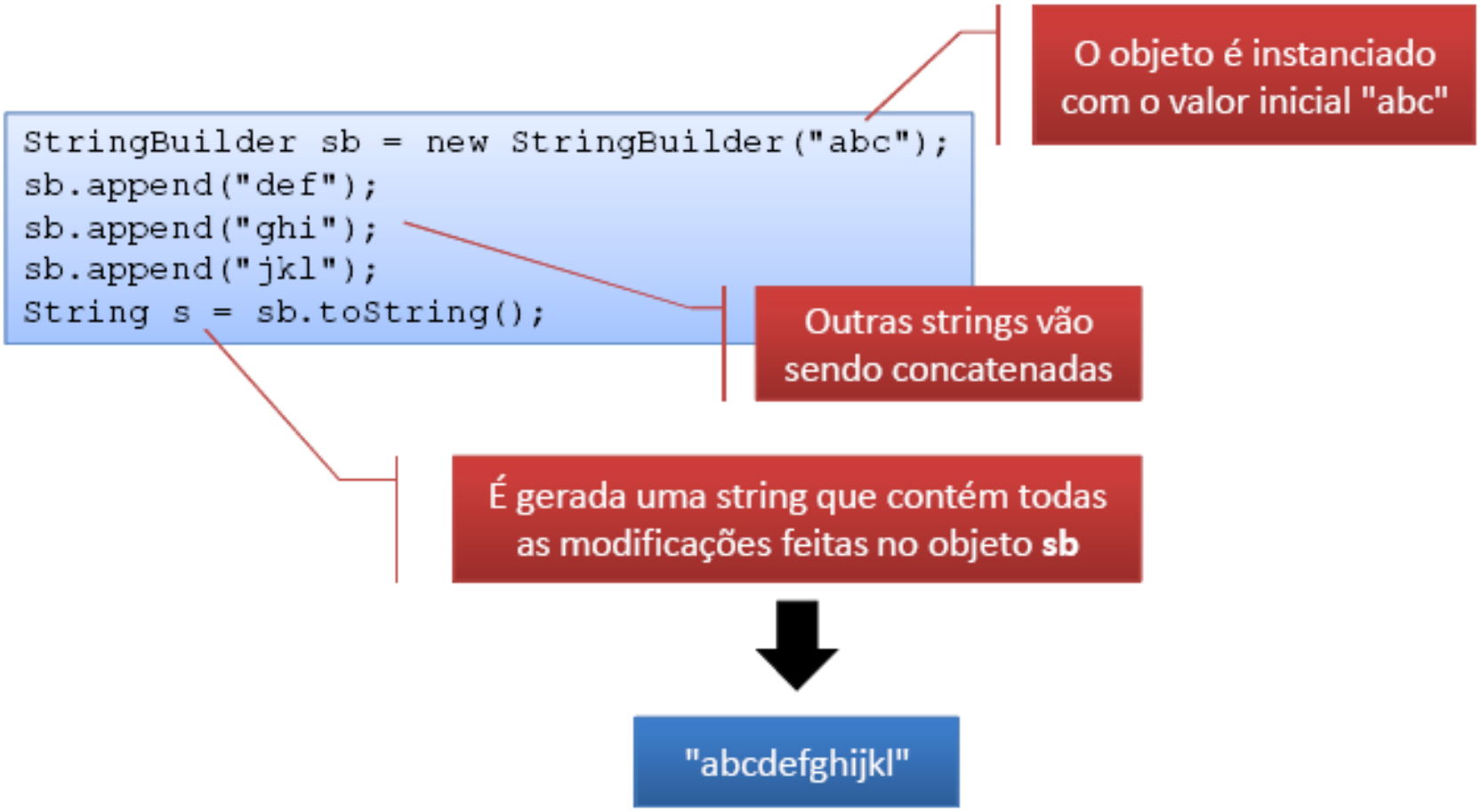
Exemplo2:

```
O tamanho da String é: 26
O caracter da posição 7 é: o
String em caixa alta:  VOAR VOAR, SORRIR SORRIR
String em caixa baixa:  voar voar, sorrir sorrir
A substring de 2 a 8 é: oar vo
Tirando os espaços em branco antes e depois temos:Voar voar, sorrir sorrir
 Voar voar, latir latir
10
12
19
```

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

Exemplo

```
public class Aplicacao {

    public static void main(String[] args) {
        /*Código carregado, criando muitas strings*/
        String s = "";
        for (int i=0; i<1000; i++) {
            s = s + "X";
        }
        System.out.println(s);

        /*Código mais eficiente:*/
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 1000; i++) {
            sb.append("Y");
        }
        String s2 = sb.toString();
        System.out.println(s2);

        /*Neste caso o java usa internamente StringBuilder*/
        System.out.println("A" + "B" + "C" + "D");
    }

}
```

Formatando Strings

- A formatação de strings pode ser feita facilmente através dos métodos format() e printf() da classe PrintStream
  - System.out é um PrintStream, portanto é possível formatar a saída para o console
- A classe String também possui o método format()

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
```

Formatando Números

- Java possui a classe NumberFormat, utilizada para formatar números
- Possui suporte à localização

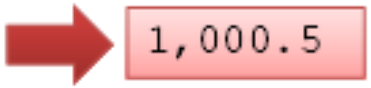
Exemplo de Formatação Numérica

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

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

- 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);
```



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
```

Exemplo

```
public class TesteNumeros {
    public static void main(String[] args) {
        double num = 1000.30;

        NumberFormat numberFormat = NumberFormat.getNumberInstance();
        System.out.println("Numero formatado padrão brasileiro: "+numberFormat.format(num));

        numberFormat = NumberFormat.getNumberInstance(new Locale("en", "us"));
        System.out.println("Numero formatado padrão estado unidense: "+numberFormat.format(num));

        NumberFormat moedaFormat = NumberFormat.getCurrencyInstance(new Locale("en", "us"));
        System.out.println("Valor em dolar: "+moedaFormat.format(num));

        moedaFormat = NumberFormat.getCurrencyInstance(new Locale("pt", "br"));
        System.out.println("Valor em real: "+moedaFormat.format(num));

        //DecimalFormat df = new DecimalFormat("00,000.000"); //caso queira separador de milhar
        DecimalFormat df = new DecimalFormat("00000.000");
        System.out.println(df.format(num));

        DecimalFormatSymbols dfs = new DecimalFormatSymbols();
        dfs.setDecimalSeparator('-');
        df.setDecimalFormatSymbols(dfs);
        System.out.println(df.format(num));
    }
}
```

Trabalhando com Datas

- Java possui quatro classes principais para trabalhar com datas

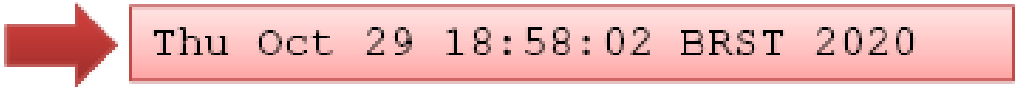
Classe	Descrição
java.util.Date	Representa uma data e hora.
java.util.Calendar	Possibilita a conversão e manipulação de datas e horas.
java.text.DateFormat	Formata datas e horas.
java.util.Locale	Representa uma localidade. É utilizada com datas para formatá-las de acordo com a localidade desejada.

- Uma nova API de datas e horas foi adicionada a partir do Java 8

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 2020
```

- Somar 7 dias à data atual

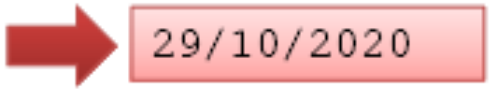
```
Calendar c = Calendar.getInstance();
c.add(Calendar.DAY_OF_MONTH, 7);
Date d = c.getTime();
System.out.println(d.toString());
```

```
Thu Nov 05 18:58:02 BRST 2020
```

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);
```



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 2020

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

API de Data e Hora: Pacote java.time

- A partir do Java 8 a linguagem conta com uma nova API para manipulação de datas e horas
- Características
  - Diversas classes para representar diferentes conceitos
  - Classes imutáveis, o que as torna thread-safe

Principais Elementos

Nome da Classe	O que representa
LocalDate	Uma data (com dia, mês e ano)
LocalTime	Uma hora (com hora, minuto, segundo e milissegundo)
LocalDateTime	Uma data e hora
Period	Um período de tempo (em anos, meses, dias, semanas)
Duration	Uma duração de tempo (em dias, horas, minutos, segundos)
MonthDay	Um par de mês e dia (Ex: dia de aniversário)
YearMonth	Um par de ano e mês (Ex: data de validade do cartão de crédito)
Instant	Um instante no tempo, com precisão de nanossegundos

Nome do Enum	O que representa
ChronoUnit	Unidades de tempo (dias, meses, anos, horas, minutos, etc.)

Classes LocalDate e LocalTime

- Data/hora atual do sistema

```
LocalDate d = LocalDate.now();
```

```
LocalTime t = LocalTime.now();
```

- Data/hora juntando as partes

```
LocalDate d = LocalDate.of(2020, Month.DECEMBER, 10);
```

```
LocalTime t = LocalTime.of(13, 45, 0);
```

- Data/hora através de parse

```
LocalDate d = LocalDate.parse("04/03/2020",
    DateTimeFormatter.ofPattern("dd/MM/yyyy"));
```

```
LocalTime t = LocalTime.parse("16:00",
    DateTimeFormatter.ofPattern("HH:mm"));
```

Operações com Datas

- LocalDate

```
LocalDate d = LocalDate.now();
LocalDate d1 = d.plusDays(5);
LocalDate d2 = d.minus(1, ChronoUnit.WEEKS);
```

- LocalTime

```
LocalTime t = LocalTime.now();
LocalTime t2 = t.plusHours(2).plusMinutes(30);
LocalTime t3 = t.minus(100, ChronoUnit.MILLIS);
```

- LocalDateTime

```
LocalDateTime d = LocalDateTime.now();
LocalDateTime d2 = d.plusDays(2).plusHours(30);
```

Classes Period e Durarion

- Período/duração juntando as partes

```
Period p = Period.of(0, 1, 7);
LocalDate d = LocalDate.now().plus(p);
```

```
Duration d = Duration.ofMinutes(15);
LocalTime t = LocalTime.now().minus(d);
```

- Período/duração entre datas datas/horas

```
LocalDate d1 = LocalDate.now();
LocalDate d2 = LocalDate.parse("2000-01-05");

Period p = Period.between(d2, d1);
int years = p.getYears();
int months = p.getMonths();
int days = p.getDays();
```

```
LocalTime t1 = LocalTime.now();
LocalTime t2 = LocalTime.parse("04:30:00");

Duration d = Duration.between(t2, t1);
long seconds = d.getSeconds();
```

Exemplo

```
String dataNascimento = "03/05/1988 05:00"; //Data Hora
LocalDateTime localDateTime1 = LocalDateTime.parse(dataNascimento, DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm"));

System.out.println("Eu nasci em "+localDateTime1);

LocalDateTime localDateTime2 = LocalDateTime.now();
System.out.println("Hoje sao: "+localDateTime2);

Duration d = Duration.between(localDateTime1, localDateTime2);

System.out.println("Eu tenho "+d.toMinutes()+" minutos vividos");
System.out.println("Eu tenho "+d.toHours()+" horas vividas");
System.out.println("Eu tenho "+d.toDays()+" dias vividos");

Period p = Period.between(localDateTime1.toLocalDate(), localDateTime2.toLocalDate());

System.out.println("Eu tenho "+p.toTotalMonths()+" meses vividos");
System.out.println("Eu tenho "+p.getYears()+" anos vividos");
System.out.println("\n\n");
```

Saída:

```
Eu nasci em 1988-05-03T05:00
Hoje sao: 2015-10-14T15:02:35.973
Eu tenho 14436602 minutos vividos
Eu tenho 240610 horas vividas
Eu tenho 10025 dias vividos
Eu tenho 329 meses vividos
Eu tenho 27 anos vividos
```

Enum ChronoUnit

- Intervalo em meses

```
LocalDate d1 = LocalDate.of(2000, Month.JANUARY, 1);
LocalDate d2 = LocalDate.of(2100, Month.DECEMBER, 31);
long months = ChronoUnit.MONTHS.between(d1, d2);
```

- Intervalo em nanossegundos

```
LocalTime t1 = LocalTime.of(8, 0);
LocalTime t2 = LocalTime.now();
long nanos = ChronoUnit.NANOS.between(t1, t2);
```

- Intervalo em horas

```
LocalTime t1 = LocalTime.of(8, 0);
LocalTime t2 = LocalTime.now();
long nanos = ChronoUnit.HOURS.between(t1, t2);
```

Exemplo

```
LocalDateTime localDateTime1 = LocalDateTime.of(1988, 5, 3, 5, 0); //03/05/1988 às 05:00h

//Formatando um LocalDate
DateTimeFormatter df = DateTimeFormatter.ofPattern("dd/MM/yyyy", new Locale("pt", "br"));
System.out.println("Eu nasci em "+df.format(localDateTime1));

LocalDateTime localDateTime2 = LocalDateTime.now();
System.out.println("Hoje sao: "+df.format(localDateTime2));

System.out.println("Eu tenho "+ChronoUnit.WEEKS.between(localDateTime1, localDateTime2)+" semanas vividas");
System.out.println("Eu tenho "+ChronoUnit.YEARS.between(localDateTime1, localDateTime2)+" anos vividos");
```

Saída:

```
Eu nasci em 03/05/1988
Hoje sao: 14/10/2015
Eu tenho 1432 semanas vividas
Eu tenho 27 anos vividos
```

Classe Instant

- Tempo de execução

```
Instant start = Instant.now();

//...

Instant end = Instant.now();

Duration d = Duration.between(start, end);
long seconds = d.getSeconds();
```

Integração com Código Legado

- As classes *Date* e *Calendar* ganharam métodos para converter a sua representação para um objeto *Instant*.

Date -> LocalDateTime

```
Date date = new Date();
Instant instant = date.toInstant();
LocalDateTime ldt = LocalDateTime.from(instant);
```

LocalDateTime -> Date

```
LocalDateTime ldt = LocalDateTime.now();
Instant instant = ldt.atZone(ZoneId.systemDefault()).toInstant();
Date date = Date.from(instant);
```

Integração com Código Legado

Calendar -> LocalDateTime

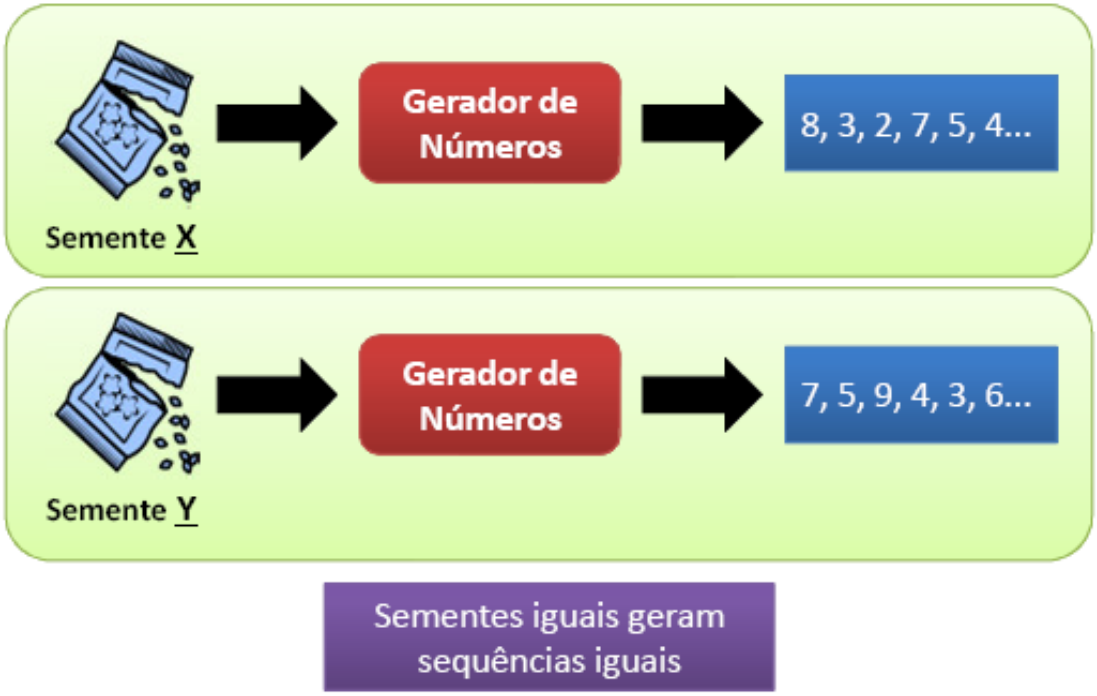
```
Calendar calendar = Calendar.getInstance();
Instant instant = calendar.toInstant();
LocalDateTime ldt = LocalDateTime.from(instant);
```

LocalDateTime -> Calendar

```
LocalDateTime ldt = LocalDateTime.now();
Instant instant = ldt.atZone(ZoneId.systemDefault()).toInstant();
Calendar calendar = Calendar.getInstance();
calendar.setTime(Date.from(instant));
```

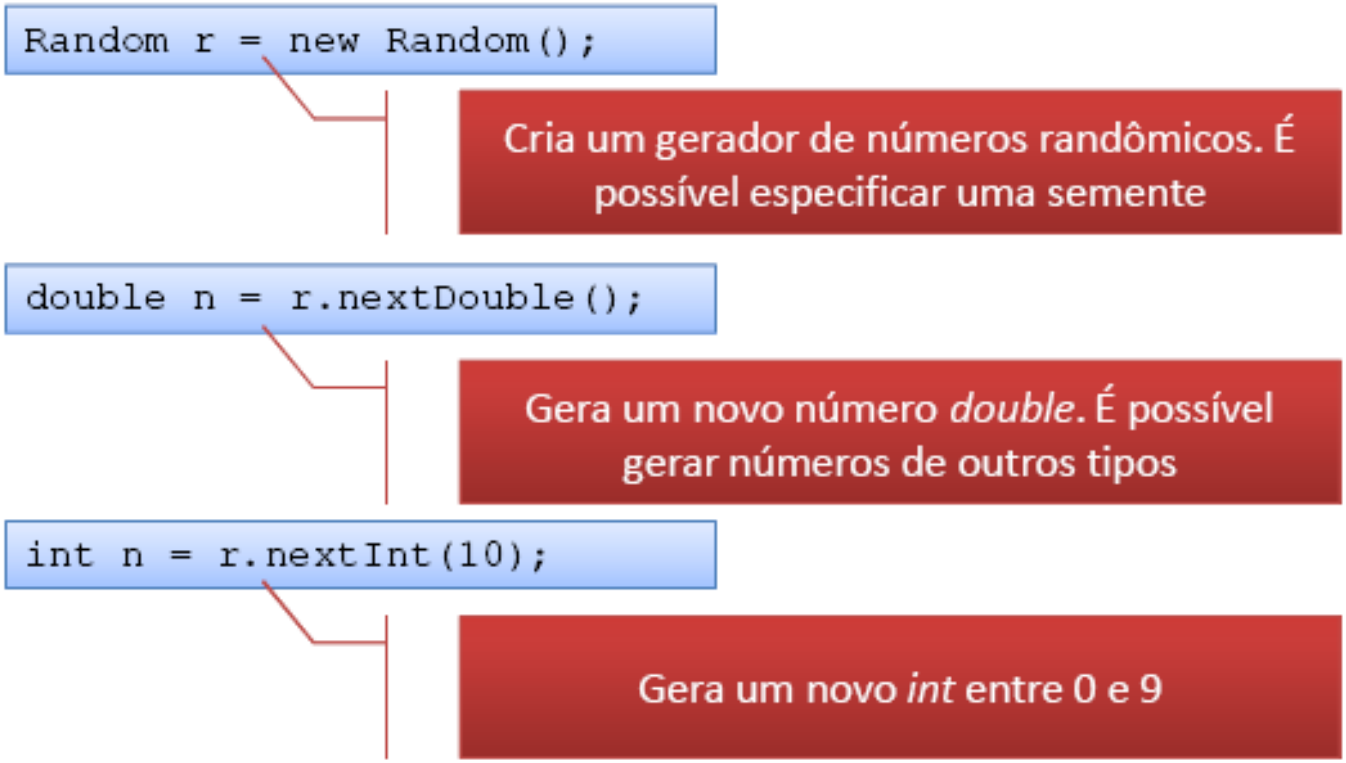
Números Randômicos

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



A Classe Random

- A classe Random pode ser utilizada para gerar números randômicos



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

Exemplo

```java
public class TesteNumeroRandomicos {
    public static void main(String[] args) {
        //Random r = new Random(10); //semente 10
        //deixando a classe escolher as sementes
        System.out.println("Usando a classe Random");
        Random r = new Random();
        for (int i=0; i<3; i++) {
            System.out.println((i+1) + "=> "+r.nextInt(100));
        }
        System.out.println("Usando o método gerar int: Intervalo[5-10]");
        for (int i=0; i<3; i++) {
            System.out.println((i+1) + "=> "+TesteNumeroRandomicos.gerarInt(5, 10));
        }
        System.out.println("Usando o método estático random da classe Math");
        for (int i=0; i<3; i++) {
            int x = (int) (Math.random()*5);
            System.out.println((i+1) + "=> "+x);
        }
    }

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

Saída

```
Usando a classe Random
1=> 31
2=> 98
3=> 44
Usando o método gerar int: Intervalo[5-10]
1=> 6
2=> 6
3=> 8
Usando o método estático random da classe Math
1=> 0
2=> 0
3=> 1
```

A Classe Math

- Possui uma série de métodos estáticos voltados para operações matemáticas comuns

Método	Descrição
abs()	Valor absoluto (sem sinal)
max()	Valor máximo entre dois valores
min()	Valor mínimo entre dois valores
ceil()	Arredonda um valor para cima
floor()	Arredonda um valor para baixo
round()	Arredonda um valor para cima ou para baixo
sqrt()	Raiz quadrada
pow()	Potenciação
toDegrees()	Ângulo de radianos para graus
toRadians()	Ângulo de graus para radianos

Links Interessantes

- Java Tutorials: Numbers and Strings
  - http://docs.oracle.com/javase/tutorial/java/data/index.html
- Java Tutorials: Formatting
  - http://docs.oracle.com/javase/tutorial/i18n/format/index.html
- Java Tutorials: Date Time
  - http://docs.oracle.com/javase/tutorial/datetime/index.html
- Java SE 8 Date and Time
  - http://www.oracle.com/technetwork/articles/java/jf14-date-time-2125367.html

Referências Bibliográficas

- DEITEL, Harvey M. e DEITEL, Paul J. Java - Como Programar, 8ª edição. Pearson. 2010.
- BLOCH, Joshua. Effective Java, 2ª edição. Addison-Wesley, 2008.
- CAELUM. Java e Orientação a Objetos. Disponível em: https://www.caelum.com.br/apostila-java-orientacao-objetos/
- SOFTBLUE. Professor Carlos Eduardo Gusso Tosin. Fundamentos de Java. http://www.softblue.com.br/.
- K19. Java e Orientação a Objetos. Disponível em: http://www.k19.com.br/cursos/orientacao-a-objetos-em-java.
- HORSTMANN, CORNELL. Core Java Volume I – Fundamentos, 8º Edição. São Paulo, Pearson Education, 2010.
- BRAUDE, E. J. Projeto de software - da programação à arquitetura: uma abordagem baseada em Java. Porto Alegre: Bookman, 2005.
- SANTOS, R. Introdução à Programação Orientada a Objetos usando Java. São Paulo: Campus, 2003.
- Slides do Professor Doutor Horácio Fernandes da UFAM.

"Péssima ideia, a de que não se pode mudar". Montaigne

filipedwan@gmail.com

;-)

skype filipedwan

twitter @filipedwan