

# Sobrecarga em Java

Dizemos que existe **sobrecarga** quando métodos de mesmo nome são visíveis em um mesmo escopo (chamada difere por tipo de parâmetros ou resultado).

Em Java, métodos sobrecarregados devem ter pelo menos um *parâmetro* de tipo diferente:

```
class ErroSobrecarga
{ int read() { ... }
  boolean read() { ... } }
```

# Classe *Console*

```
static boolean  readBoolean()  
static double  readByte()  
static short   readShort()  
static int     readInt()  
static long    readLong()  
static float   readFloat()  
static double  readDouble()  
static boolean readBoolean()  
static String readString()
```



## Classe *BufferedReader*

- Contém fonte de caracteres, de onde os caracteres são de fato lidos.
- Métodos (ex: *readLine*) usam áreas de armazenamento temporário (“buffers”) para melhorar o desempenho das operações de entrada de dados.
- Construtor recebe como parâmetro objeto (fonte de caracteres) da classe *InputStreamReader*.

# Classe *Console*: implementação

```
import java.io.*;
public class Console
{final static BufferedReader console =
    new BufferedReader(new InputStreamReader(System.in));
final static PrintStream terminal = System.out;

...
}
```

```
public static byte readByte()
{try { return Byte.parseByte(console.readLine());
  catch(IOException e)
    { terminal.println("IOException: tente de novo.");
      return readByte(); }
  catch(NumberFormatException e)
    { terminal.println("NumberFormatException \n" +
      "Valor entre -128 e 127 esperado: digite novamente");
      return readByte(); } }
```

```
public static String readString ()
{ try    { return console.readLine(); }
  catch (IOException e)
        { terminal.println( "IOException: tente de novo.");
          return readString(); }
}
```



## Classe *StreamTokenizer*

- *StreamTokenizer* provê suporte a *análise léxica*.
- **Análise léxica:** separação da entrada (contendo uma cadeia de caracteres) em *elementos léxicos* (“*tokens*”), descartando *caracteres delimitadores* (como espaços, caracteres de tabulação e de mudança de linha).
- Classe com comportamento típico de POO (métodos mudam *estado*, que pode então ser “consultado”).

## Método *nextToken*

Método *nextToken* de *StreamTokenizer* lê “próximo” elemento léxico de uma fonte (“*stream*”) de caracteres (objeto do tipo *StreamTokenizer*) e modifica variáveis desse objeto, de acordo com elemento léxico lido.

Se for lido	Modifica	de Tipo
valor numérico	variável <i>nval</i> (e <i>ttype</i> )	<code>double</code>
identificador	variável <i>sval</i> (e <i>ttype</i> )	<code>String</code>
caso contrário	variável <i>ttype</i>	<code>int</code>

# Variável *ttype*

*nextToken* retorna e armazena em *ttype* tipo do elemento léxico lido:

- *TT\_NUMBER* — indica que foi lido valor numérico;
- *TT\_WORD* — indica que foi lido um identificador;
- *TT\_EOF* — indica fim da entrada de caracteres;
- *TT\_EOL* — indica fim de linha;
- nenhuma das opções acima — tipo do elemento léxico é o valor da representação do caractere. Ex: se '\*' for lido, valor retornado e armazenado em *ttype* é igual ao código Unicode de '\*'.

# Especificando identificadores

- *identificador* caracterizado por: *ttype* = *TT\_WORD*
- Usualmente, iniciado com letra (ou outro caractere que indica início de um nome) e seguido por letras e dígitos
- *wordChars*, definido em *StreamTokenizer*, especifica caracteres de identificadores:

```
public void wordChars (int c1, int c2)
```

- caracteres com código na faixa de *c1* a *c2* passam a ser considerados como componentes de identificadores

# Especificando delimitadores

- `public void whiteSpaceChars(int c1, int c2)`
  - Define quais caracteres são delimitadores (separam identificadores), de maneira análoga a `wordChars`
- `public void eolIsSignificant(boolean b)`
  - Habilita ou desabilita atribuição de `TT_EOL` a `ttype`
- `public void resetSyntax()`
  - Especifica que nenhum caractere é reconhecido como identificador ou delimitador.

## Usando *StreamTokenizer*: exemplo

Programa a seguir usa *StreamTokenizer* para contar número de caracteres, palavras e linhas na entrada padrão.

```
import java.io.*;

class ContaPalavras
{ public static int palavras = 0;
  public static int linhas = 0;
  public static int caracteres = 0;
  ...
}
```

```
public static void conta_palavras (InputStreamReader r)
    throws IOException
{ StreamTokenizer st = new StreamTokenizer(r);
  st.resetSyntax(); st.eolIsSignificant(true);
  st.wordChars(33, 255); st.whitespaceChars(0, ' ');
  while (st.nextToken() != st.TT_EOF)
    switch (st.ttype) {
      case st.TT_EOL: linhas++; break;
      case st.TT_WORD: palavras++;
        caracteres += st.sval.length(); break;
      default: caracteres += st.sval.length(); break;
    } }
```

```
public static void main (String[] args)
{ try { conta_palavras(new InputStreamReader(System.in));
      System.out.println("\n" +
                          "linhas = "      + linhas      + "\n" +
                          "palavras = "    + palavras    + "\n" +
                          "caracteres = "  + caracteres); }
  catch (IOException e) { };
}
```