Strings

UA.DETI.POO

A classe String

- A classe java.lang.String facilita a manipulação de cadeias de carateres.
- Exemplo:

```
String s1 = "java"; // creating string by java string literal
char ch[] = { 's', 't', 'r', 'i', 'n', 'g', 's' };
String s2 = new String(ch); // converting char array to string
System.out.println(s1);
System.out.println(s2);
```

```
java
strings
```

Concatenação de Strings

- Concatenação de Strings

```
String data = " feve" + "reiro ";
data = 10 + data;
data += "de " + 2019;
System.out.println(data);
```

- Os objetos to tipo String são imutáveis (constantes).
  - Todos os métodos cujo objetivo é modificar uma String, na realidade constroem e devolvem uma String nova
  - A String original mantém-se inalterada.

  - Quantos objetos String existem no código acima?

Concatenação de Strings

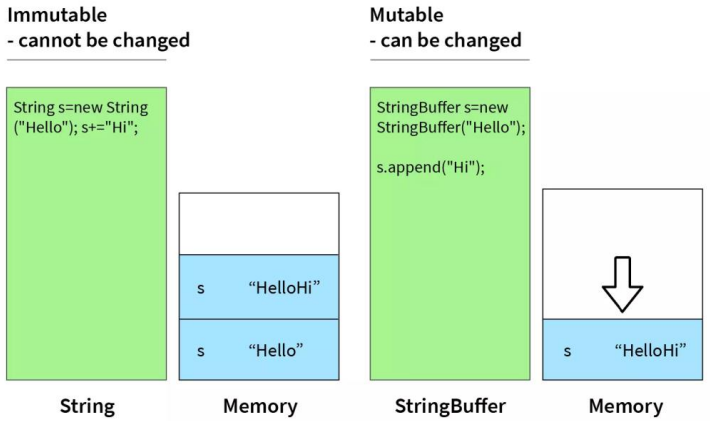
- Utilização alternativa do tipo StringBuilder

```
StringBuilder sb = new StringBuilder();
sb.append(10);
sb.append(" feve");
sb.append("reiro ");
sb.append("de ");
sb.append(2019);
String data = sb.toString();
System.out.println(data);
```

```
10 fevereiro de 2019
```

String vs StringBuilder

Features	String	StringBuilder
Mutability	String are immutable(creates new objects on modification)	StringBuilder are mutable(modifies in place)
Thread-Safe	It is thread-safe	It is not thread-safe
Performance	It is slow because it creates an object each time	It is faster (no object creation)
use Case	Fixed, unchanging strings	Single-threaded string manipulatio

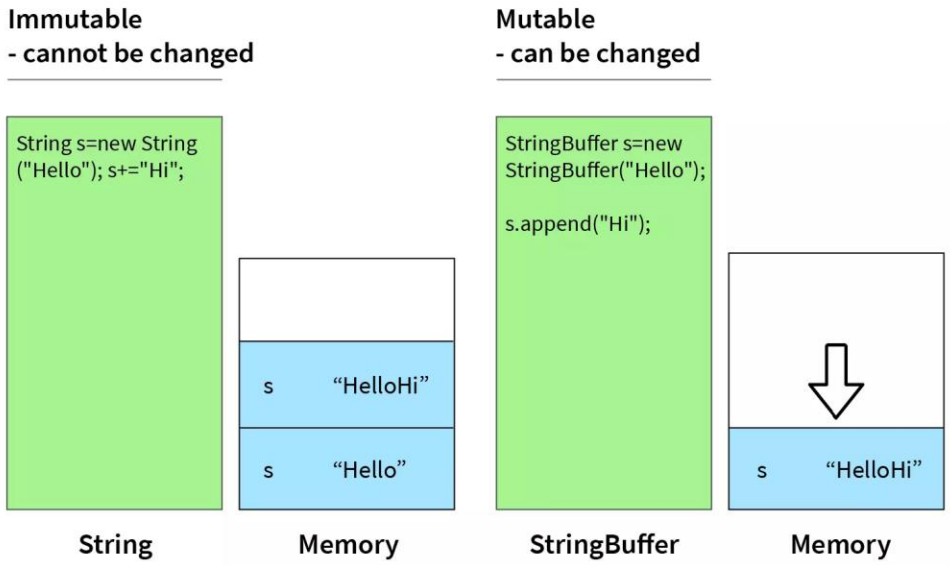


Conclusion

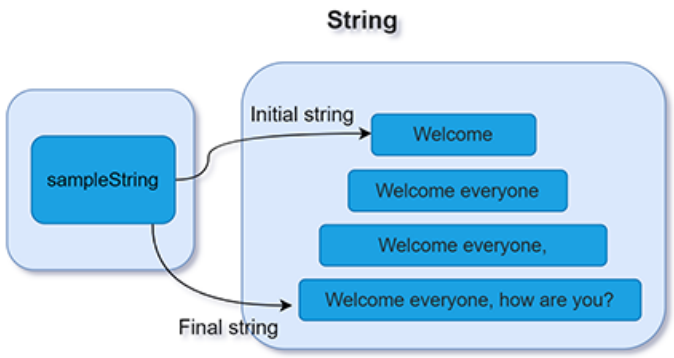
The choice between `String` and `StringBuilder` in Java depends on your specific requirements. For simple and small-scale operations where thread safety is required, `String` is a good choice. However, for large-scale or performance-critical operations involving numerous manipulations, `StringBuilder` is typically the better option.

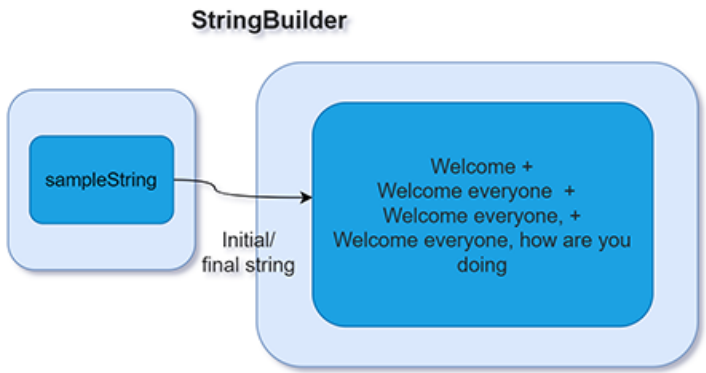
https://www.geeksforgeeks.org/java/string-vs-stringbuilder-vs-stringbuffer-in-java/
https://medium.com/@AlexanderObregon/understanding-string-vs-stringbuilder-in-java-50448cbbf253
https://javapapers.com/java/java-string-vs-stringbuilder-vs-stringbuffer-concatenation-performance-micro-benchmark/

String vs StringBuilder



SCALER Topics

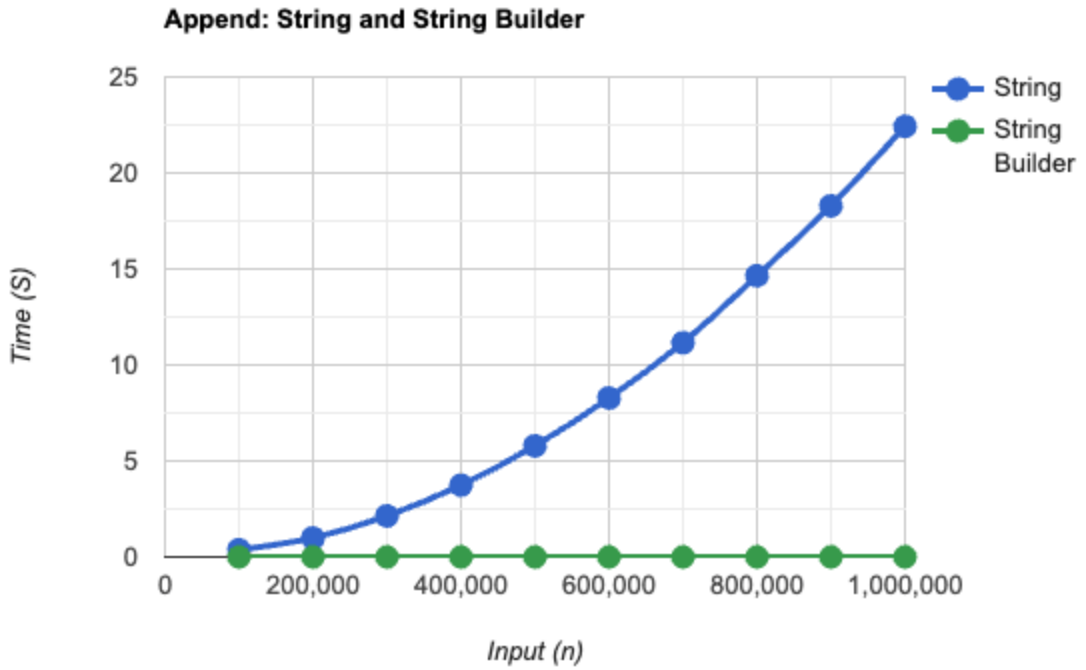




https://dip-mazumder.medium.com/stringbuilder-vs-string-in-java-a-guide-for-optimal-memory-usage-4a284d8243ea

String vs StringBuilder

```
String str1 = "Hello";
String str2 = str1.concat(", World"); // Creates a new
String
System.out.println(str1); // Output: Hello
System.out.println(str2); // Output: Hello, World
```




```
StringBuilder stringBuilder = new StringBuilder("Hello");
stringBuilder.append(", World"); // Modifies the
StringBuilder in place
String result = stringBuilder.toString(); // Converts to
String
System.out.println(result); // Output: Hello, World
```

https://dip-mazumder.medium.com/stringbuilder-vs-string-in-java-a-guide-for-optimal-memory-usage-4a284d8243ea

Métodos da class String

- Esta classe apresenta um conjunto de métodos que permitem realizar muitas operações sobre texto.

char	**charAt**(int index) Returns the char value at the specified index.
int	**codePointAt**(int index) Returns the character (Unicode code point) at the specified index.
int	**codePointBefore**(int index) Returns the character (Unicode code point) before the specified index.
int	**codePointCount**(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
int	**compareTo**(**String** anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(**String** str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(**String** str) Concatenates the specified string to the end of this string.
boolean	**contains**(**CharSequence** s) Returns true if and only if this string contains the specified sequence of char values.

https://www.w3schools.com/java/java_strings.asp
https://www.tutorialspoint.com/java/lang/java_lang_string.htm

Comprimento e acesso a carateres

- O comprimento (número de carateres) de uma String pode ser determinado com o método length.
- O acesso a um carater é feito com o método charAt (int index).
- Exemplo:

```
String s1 = "Universidade de Aveiro";
System.out.println(s1.length());
for (int i=0; i < s1.length(); i++ )
    System.out.print(s1.charAt(i) + ", ");
```

```
22
U, n, i, v, e, r, s, i, d, a, d, e,  , d, e,  , A, v, e, i, r, o,
```

Comparação de Strings

- Alguns métodos
  - equals, equalsIgnoreCase, compareTo

- Exemplos:

```
String s1 = "Aveiro";
String s2 = "aveiro";

System.out.println(s1.equals(s2) ? "Iguais" : " Diferentes");
System.out.println(s1.equalsIgnoreCase(s2) ? "Iguais" : " Diferentes ");

System.out.println(s1.compareTo(s2));

    // <0 (s1 menor), 0(iguais), >0 (s1 maior)
```

Comparação de subStrings

- Podemos analisar partes de uma String
  - contains, substring, startsWith, endsWith, …

- Exemplos:

```
String s1 = "Aveiro";
String s2 = "aveiro";

// is part of ?
System.out.println(s1.contains("ve"));     // true
System.out.println(s1.startsWith("ave")); // false
System.out.println(s1.endsWith("ro"));     // true

// get part of
System.out.println(s1.substring(1, 3));    // ve
```

Formatação de Strings

- O método format retorna uma String nova formatada de acordo com especificadores de formato.

```
long segundos = 347876;
String s1 =
String.format("%02d horas, %02d minutos e %02d segundos\n",
    segundos / 3600,
    (segundos % 3600) / 60,
    segundos % 60);
System.out.println(s1);
```

```
96 horas, 37 minutos e 56 segundos
```

https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

Formatação de Strings

- O método format retorna uma String nova formatada de acordo com especificadores de formato.

```
long segundos = 347876;
String s1 =
String.format("%02d horas, %02d minutos e %02d segundos\n",
    segundos / 3600,
    (segundos % 3600) / 60,
    segundos % 60);
System.out.println(s1);
```

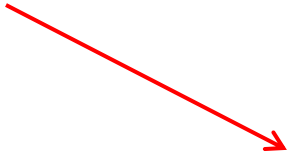
```
96 horas, 37 minutos e 56 segundos
```

https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

Formatação de Strings

- O método format retorna uma String nova formatada de acordo com especificadores de formato.

```
long segundos = 347876;
String s1 =
String.format("%02d horas, %02d minutos e %02d segundos\n",
    segundos / 3600,
    (segundos % 3600) / 60,
    segundos % 60);
System.out.println(s1);
```



```
96 horas, 37 minutos e 56 segundos
```

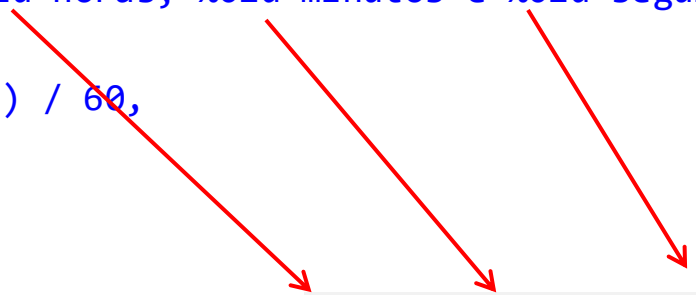
https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

Formatação de Strings

- System.out.printf é um método, alternativo ao System.out.print, que utiliza formatação.

- Exemplo:

```
long segundos = 347876;
System.out.printf("%02d horas, %02d minutos e %02d segundos\n",
      segundos / 3600,
      (segundos % 3600) / 60,
      segundos % 60);
```



```
96 horas, 37 minutos e 56 segundos
```

https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

Expressões regulares

motivation

- read a tab separated file containing students names and grades from 3 tests and present average

```
Name    Test1   Test2   Test3
Alice   85      90      88
Bob     70      75      72
Charlie 92      89      95
Diana   60      65      63
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadStudents {
    public static void main(String[] args) {
        String filename = "students.tsv";

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // read header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                String[] parts = line.split("\t");

                String name = parts[0];
                int test1 = Integer.parseInt(parts[1]);
                int test2 = Integer.parseInt(parts[2]);
                int test3 = Integer.parseInt(parts[3]);

                double avg = (test1 + test2 + test3) / 3.0;

                System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                        name, test1, test2, test3, avg);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

```python
import csv

filename = "students.tsv"

with open(filename, "r", newline="") as file:
    reader = csv.reader(file, delimiter="\t")

    header = next(reader)  # skip header
    print(header)

    for row in reader:
        name = row[0]
        test1 = int(row[1])
        test2 = int(row[2])
        test3 = int(row[3])

        avg = (test1 + test2 + test3) / 3
        print(f"{name}: scores = {test1}, {test2}, {test3} | average = {avg:.2f}")
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadStudents {
    public static void main(String[] args) {
        String filename = "students.tsv";

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // read header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                String[] parts = line.split("\t");

                String name = parts[0];
                int test1 = Integer.parseInt(parts[1]);
                int test2 = Integer.parseInt(parts[2]);
                int test3 = Integer.parseInt(parts[3]);

                double avg = (test1 + test2 + test3) / 3.0;

                System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                        name, test1, test2, test3, avg);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

```python
import csv

filename = "students.tsv"

with open(filename, "r", newline="") as file:
    reader = csv.reader(file, delimiter="\t")

    header = next(reader)  # skip header
    print(header)

    for row in reader:
        name = row[0]
        test1 = int(row[1])
        test2 = int(row[2])
        test3 = int(row[3])

        avg = (test1 + test2 + test3) / 3
        print(f"{name}: scores = {test1}, {test2}, {test3} | average = {avg:.2f}")
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ReadStudentsRegex {
    public static void main(String[] args) {
        String filename = "students.tsv";

        // Regex: Name<TAB>num<TAB>num<TAB>num
        Pattern pattern = Pattern.compile("^(.+?)\\t(\\d+)\\t(\\d+)\\t(\\d+)$");

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                Matcher matcher = pattern.matcher(line);

                if (matcher.matches()) {
                    String name = matcher.group(1);
                    int test1 = Integer.parseInt(matcher.group(2));
                    int test2 = Integer.parseInt(matcher.group(3));
                    int test3 = Integer.parseInt(matcher.group(4));

                    double avg = (test1 + test2 + test3) / 3.0;

                    System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                            name, test1, test2, test3, avg);
                } else {
                    System.out.println("Invalid line: " + line);
                }
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadStudents {
    public static void main(String[] args) {
        String filename = "students.tsv";

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // read header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                String[] parts = line.split("\t");

                String name = parts[0];
                int test1 = Integer.parseInt(parts[1]);
                int test2 = Integer.parseInt(parts[2]);
                int test3 = Integer.parseInt(parts[3]);

                double avg = (test1 + test2 + test3) / 3.0;

                System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                        name, test1, test2, test3, avg);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ReadStudentsRegex {
    public static void main(String[] args) {
        String filename = "students.tsv";

        // Regex: Name<TAB>num<TAB>num<TAB>num
        Pattern pattern = Pattern.compile("^(.+?)\\t(\\d+)\\t(\\d+)\\t(\\d+)$");

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                Matcher matcher = pattern.matcher(line);

                if (matcher.matches()) {
                    String name = matcher.group(1);
                    int test1 = Integer.parseInt(matcher.group(2));
                    int test2 = Integer.parseInt(matcher.group(3));
                    int test3 = Integer.parseInt(matcher.group(4));

                    double avg = (test1 + test2 + test3) / 3.0;

                    System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                            name, test1, test2, test3, avg);
                } else {
                    System.out.println("Invalid line: " + line);
                }
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadStudents {
    public static void main(String[] args) {
        String filename = "students.tsv";

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // read header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                String[] parts = line.split("\t");

                String name = parts[0];
                int test1 = Integer.parseInt(parts[1]);
                int test2 = Integer.parseInt(parts[2]);
                int test3 = Integer.parseInt(parts[3]);

                double avg = (test1 + test2 + test3) / 3.0;

                System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                        name, test1, test2, test3, avg);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

motivation

- read a tab separated file containing students names and grades from 3 tests and present average

```
Name    Test1   Test2   Test3
Alice   85      90      88
Bob     70      75      72
Charlie 92      89      95
Diana   60      65      63
```

```
// Regex: Name<TAB>num<TAB>num<TAB>num
Pattern pattern =Pattern.compile("^(.+?)\\t(\\d+)\\t(\\d+)\\t(\\d+)$");
```

motivation

- read a tab separated file containing students names and grades from 3 tests and present average

```
Name    Test1   Test2   Test3
Alice   85      90      88
Bob     70      75      72
Charlie 92      89      95
Diana   60      65      63
```

```
// Regex: Name<TAB>num<TAB>num<TAB>num
Pattern pattern =Pattern.compile("^(.+?)\\t(\\d+)\\t(\\d+)\\t(\\d+)$");
```

start

end

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ReadStudentsRegex {
    public static void main(String[] args) {
        String filename = "students.tsv";

        // Regex: Name<TAB>num<TAB>num<TAB>num
        Pattern pattern = Pattern.compile("^(.+?)\\t(\\d+)\\t(\\d+)\\t(\\d+)$");

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                Matcher matcher = pattern.matcher(line);

                if (matcher.matches()) {
                    String name = matcher.group(1);
                    int test1 = Integer.parseInt(matcher.group(2));
                    int test2 = Integer.parseInt(matcher.group(3));
                    int test3 = Integer.parseInt(matcher.group(4));

                    double avg = (test1 + test2 + test3) / 3.0;

                    System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                            name, test1, test2, test3, avg);
                } else {
                    System.out.println("Invalid line: " + line);
                }
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadStudents {
    public static void main(String[] args) {
        String filename = "students.tsv";

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // read header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                String[] parts = line.split("\t");

                String name = parts[0];
                int test1 = Integer.parseInt(parts[1]);
                int test2 = Integer.parseInt(parts[2]);
                int test3 = Integer.parseInt(parts[3]);

                double avg = (test1 + test2 + test3) / 3.0;

                System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                        name, test1, test2, test3, avg);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ReadStudentsRegex {
    public static void main(String[] args) {
        String filename = "students.tsv";

        // Regex: Name<TAB>num<TAB>num<TAB>num
        Pattern pattern = Pattern.compile("^(.+?)\\t(\\d+)\\t(\\d+)\\t(\\d+)$");

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                Matcher matcher = pattern.matcher(line);

                if (matcher.matches()) {
                    String name = matcher.group(1);
                    int test1 = Integer.parseInt(matcher.group(2));
                    int test2 = Integer.parseInt(matcher.group(3));
                    int test3 = Integer.parseInt(matcher.group(4));

                    double avg = (test1 + test2 + test3) / 3.0;

                    System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                            name, test1, test2, test3, avg);
                } else {
                    System.out.println("Invalid line: " + line);
                }
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

Ensures the “group” are according to the pattern

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadStudents {
    public static void main(String[] args) {
        String filename = "students.tsv";

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // read header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                String[] parts = line.split("\t");

                String name = parts[0];
                int test1 = Integer.parseInt(parts[1]);
                int test2 = Integer.parseInt(parts[2]);
                int test3 = Integer.parseInt(parts[3]);

                double avg = (test1 + test2 + test3) / 3.0;

                System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                        name, test1, test2, test3, avg);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ReadStudentsRegex {
    public static void main(String[] args) {
        String filename = "students.tsv";

        // Regex: Name<TAB>num<TAB>num<TAB>num
        Pattern pattern = Pattern.compile("^(.+?)\\t(\\d+)\\t(\\d+)\\t(\\d+)$");

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                Matcher matcher = pattern.matcher(line);

                if (matcher.matches()) {
                    String name = matcher.group(1);
                    int test1 = Integer.parseInt(matcher.group(2));
                    int test2 = Integer.parseInt(matcher.group(3));
                    int test3 = Integer.parseInt(matcher.group(4));

                    double avg = (test1 + test2 + test3) / 3.0;

                    System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                            name, test1, test2, test3, avg);
                } else {
                    System.out.println("Invalid line: " + line);
                }
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadStudents {
    public static void main(String[] args) {
        String filename = "students.tsv";

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // read header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                String[] parts = line.split("\t");

                String name = parts[0];
                int test1 = Integer.parseInt(parts[1]);
                int test2 = Integer.parseInt(parts[2]);
                int test3 = Integer.parseInt(parts[3]);

                double avg = (test1 + test2 + test3) / 3.0;

                System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                        name, test1, test2, test3, avg);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

Trust that there are 4 columns split by “\t”…
If not, you get an error

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ReadStudentsRegex {
    public static void main(String[] args) {
        String filename = "students.tsv";

        // Regex: Name<TAB>num<TAB>num<TAB>num
        Pattern pattern = Pattern.compile("^(.+?)\\t(\\d+)\\t(\\d+)\\t(\\d+)$");

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                Matcher matcher = pattern.matcher(line);

                if (matcher.matches()) {
                    String name = matcher.group(1);
                    int test1 = Integer.parseInt(matcher.group(2));
                    int test2 = Integer.parseInt(matcher.group(3));
                    int test3 = Integer.parseInt(matcher.group(4));

                    double avg = (test1 + test2 + test3) / 3.0;

                    System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                            name, test1, test2, test3, avg);
                } else {
                    System.out.println("Invalid line: " + line);
                }
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadStudents {
    public static void main(String[] args) {
        String filename = "students.tsv";

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line = br.readLine(); // read header
            System.out.println(line);

            while ((line = br.readLine()) != null) {
                String[] parts = line.split("\t");

                String name = parts[0];
                int test1 = Integer.parseInt(parts[1]);
                int test2 = Integer.parseInt(parts[2]);
                int test3 = Integer.parseInt(parts[3]);

                double avg = (test1 + test2 + test3) / 3.0;

                System.out.printf("%s: scores = %d, %d, %d | average = %.2f%n",
                        name, test1, test2, test3, avg);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

String matches - regexp

- podem ser procurados em Strings.
  - A lista completa de construções suportadas está descrita na documentação da classe java.util.regex.Pattern.
- O método matches da classe String
  - verifica se uma String inclui um dado padrão.
- Exemplos:

```
String s1 = "123";
System.out.println(s1.matches("\\d{2,4}"));        // 2-4 dígitos seguidos

s1 = "abcdefg";
System.out.println(s1.matches("\\w{3,}"));         // pelo menos 3 carateres alfanuméricos
```

```
true
true
```

Alguns exemplos de padrões regex

- . qualquer caracter
- \d dígito de 0 a 9
- \D não dígito [^0-9]
- \s "espaço": [\t\n\x0B\f\r]
- \S não "espaço": [^\s]
- \w carater alfanumérico: [a-zA-Z_0-9]
- \W carater não alfanumérico: [^\w]
- [abc] qualquer dos carateres a, b ou c
- [^abc] qualquer carater exceto a, b e c
- [a-z] qualquer carater entre a-z, inclusive
- X? um ou nenhum X
- X* nenhum ou vários X
- X+ um ou vários X

String matches - regexp

- podem ser procurados em Strings.
  - A lista completa de construções suportadas está descrita na documentação da classe java.util.regex.Pattern.
- O método matches da classe String
  - verifica se uma String inclui um dado padrão.
- Exemplos:

```
String s1 = "123";
System.out.println(s1.matches("\\d{2,4}"));          // 2-4 dígitos seguidos

s1 = "abcdefg";
System.out.println(s1.matches("\\w{3,}"));           // pelo menos 3 carateres alfanuméricos
```

```
true
true
```

String matches - regexp

- podem ser procurados em Strings.
  - A lista completa de construções suportadas está descrita na documentação da classe java.util.regex.Pattern.
- O método matches da classe String
  - verifica se uma String inclui um dado padrão.
- Exemplos:

```
String s1 = "123";
System.out.println(s1.matches("\\d{2,4}"));          // 2-4 dígitos seguidos

s1 = "abcdefg";
System.out.println(s1.matches("\\w{3,}"));           // pelo menos 3 carateres alfanuméricos
```

```
true
true
```

String matches - regexp

- podem ser procurados em Strings.
  - A lista completa de construções suportadas está
    classe java.util.regex.Pattern.
- O método matches da classe String
  - verifica se uma String inclui um dado padrão.
- Exemplos:

```
.          qualquer caracter
\d   dígito de 0 a 9
\D   não dígito [^0-9]
\s   "espaço": [ \t\n\x0B\f\r]
\S   não "espaço": [^\s]
\w   carater alfanumérico: [a-zA-Z_0-9]
\W   carater não alfanumérico: [^\w]
[abc]          qualquer dos carateres a,
b ou c
[^abc]         qualquer carater exceto
a, b e c
[a-z]          qualquer carater entre a-
z, inclusive
X?   um ou nenhum X
X*   nenhum ou vários X
X+   um ou vários X
```

```
String s1 = "123";
System.out.println(s1.matches("\\d{2,4}"));        // 2-4 dígitos seguidos

s1 = "abcdefg";
System.out.println(s1.matches("\\w{3,}"));         // pelo menos 3 carateres alfanuméricos
```

```
true
true
```

String split

- O método split separa uma String em partes com base numa expressão regular e devolve o vetor de Strings resultantes.

```
String frase = "Regular expressions are powerful and "
                   + "flexible text-processing tools.";
// separa com base em carateres não alfanuméricos
String[] splitResult = frase.split("\\W");
System.out.println(splitResult.length + " palavras: "+
        Arrays.toString(splitResult));

// separa com base em “ex”
splitResult = frase.split("ex");
System.out.println(splitResult.length + " palavras: " +
                        Arrays.toString(splitResult));
```

```
9 palavras: [Regular, expressions, are, powerful, and, flexible, text, processing, tools]
4 palavras: [Regular , pressions are powerful and fl, ible t, t-processing tools.]
```

String split

- O método split separa uma String em partes com base numa expressão regular e devolve o vetor de Strings resultantes.

```
String frase = "Regular expressions are powerful and "
                    + "flexible text-processing tools.";
// separa com base em carateres não alfanuméricos
String[] splitResult = frase.split("\\W");
System.out.println(splitResult.length + " palavras: "
        Arrays.toString(splitResult));

// separa com base em "ex"
splitResult = frase.split("ex");
System.out.println(splitResult.length + " palavras: "
                        Arrays.toString(splitResult));
```

```
.           qualquer caracter
\d   dígito de 0 a 9
\D   não dígito [^0-9]
\s   "espaço": [ \t\n\x0B\f\r]
\S   não "espaço": [^\s]
\w   carater alfanumérico: [a-zA-Z_0-9]
\W   carater não alfanumérico: [^\w]
[abc]         qualquer dos carateres a, b ou c
[^abc]        qualquer carater exceto a, b e c
[a-z]         qualquer carater entre a-z,
inclusive
X?   um ou nenhum X
X*   nenhum ou vários X
X+   um ou vários X
```

```
9 palavras: [Regular, expressions, are, powerful, and, flexible, text, processing, tools]
4 palavras: [Regular , pressions are powerful and fl, ible t, t-processing tools.]
```

Some examples

Username (3–16 chars)	`^[A-Za-z0-9_]{3,16}$`	`user_123`
Strong password	`^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[@$!%*?&])[A-Za-z\\d@$!%*?&]{8,}$`	`Abc@1234`
Hex color	`^#([A-Fa-f0-9]{6})$`	`#12AB9F`
Credit card (basic)	`^(\\d{4}[- ]?){3}\\d{4}$`	`1234-5678-9012-3456`
UUID	`^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}$`	`550e8400-e29b-41d4-a716-446655440000`
Hashtag	`^#[A-Za-z0-9_]+$`	`#JavaRegex`
Java class name	`^[A-Z][A-Za-z0-9]*$`	`StudentRecord`
Java identifier	`^[A-Za-z_$][A-Za-z\\d_$]*$`	`_count2`
HTML tag (simple)	`<[^>]+>`	`<p>` / `<div class="x">`
Remove HTML tags (replace)	`<[^>]*>`	removes `<b>Hi</b>`
Whitespace (any)	`\\s+`	matches `" \n\t"`
Trim leading spaces	`^\\s+`	`" hello"`
Trim trailing spaces	`\\s+$`	`"hello "`
Multiple spaces → one	`\\s{2,}`	`"a b"`
Extract words	`[A-Za-z]+`	`"Hello world"`
Extract numbers	`\\d+`	`"ID=12345"`
File extension	`^.+\\.(\\w+)$`	`photo.png`
Basic log line (LEVEL)	`` `^(INFO ``	WARN

Some examples

Pattern Type	Java Regex (Copy/Paste)	Example Match
Email (simple)	`^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$`	bob@gmail.com
Email (better)	`^[A-Za-z0-9+_.-]+@[A-Za-z0-9-]+(\\.[A-Za-z0-9-]+)+$`	bob@mail.co.uk
HTTP / HTTPS URL	`^(https?://)([\\w-]+\\.)+[\\w-]+(/[\\w-./?%&=]*)?$`	https://example.com/a?q=1
Domain name	`^([a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,}$`	openai.com
IPv4 Address	`^((25[0-5]|2[0-4]\\d|1\\d\\d|[1-9]?\\d)\\.){3}(25[0-5]|2[0-4]\\d|1\\d\\d|[1-9]?\\d)$`	192.168.1.1
Phone (simple international)	`^\\+?[0-9]{7,15}$`	+14155552671
US Phone format	`^(\\(\\d{3}\\)|\\d{3})[- ]?\\d{3}[- ]?\\d{4}$`	(123) 456-7890
ZIP code (US)	`^\\d{5}(-\\d{4})?$`	90210 / 90210-1234
Date (YYYY-MM-DD)	`^\\d{4}-\\d{2}-\\d{2}$`	2026-02-09
Time (24h HH:MM)	`^([01]\\d|2[0-3]):[0-5]\\d$`	23:45
Integer number	`^-?\\d+$`	-42
Decimal number	`^-?\\d+(\\.\\d+)?$`	-3.14
Letters only	`^[A-Za-z]+$`	HelloWorld
Alphanumeric only	`^[A-Za-z0-9]+$`	abc123

Some examples

Pattern Type	Java Regex (Copy/Paste)	Example Match
Email (simple)	`^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$`	`bob@gmail.com`
Email (better)	`^[A-Za-z0-9+_.-]+@[A-Za-z0-9-]+(\\.[A-Za-z0-9-]+)+$`	`bob@mail.co.uk`
HTTP / HTTPS URL	`^(htt`	
Domain name	`^([a-`	
IPv4 Address	`^((25`	
Phone (simple international)	`^\\+?`	
US Phone format	`^(\\(`	
ZIP code (US)	`^\\d{`	
Date (YYYY-MM-DD)	`^\\d{`	
Time (24h HH:MM)	`^([01`	
Integer number	`^-?\\`	
Decimal number	`^-?\\`	
Letters only	`^[A-Z`	
Alphanumeric only	`^[A-Z`	

You have this in python and the patterns are similar
(we did not address it in FP)

```python
import re

pattern = re.compile(r"^\d{4}-\d{2}-\d{2}$")
print(bool(pattern.match("2026-02-09")))
```

Yep! 😄 Python regex is almost the same — the main difference is:

- In **Java** you write: `\\d`
- In **Python raw strings** you write: `r"\d"`