**Ler algo do teclado**

**private** **static** **void** lerTeclado() {

Scanner keyboard = **new** Scanner(System.***in***);

System.***out***.println("digita algo: ");

String text = keyboard.nextLine();

System.***out***.println(text);

System.***out***.println("digite um numero: ");

Integer inteiro = keyboard.nextInt();

System.***out***.println("" + inteiro);

}

**Criando um .jar executável**

* <http://pt.wikihow.com/Criar-um-Arquivo-Execut%C3%A1vel-no-Eclipse>
* <http://launch4j.sourceforge.net/>

**Strings**

* Concatenar com + ou StringBuilder ou StringBuffer
  + (<http://www.javapractices.com/topic/TopicAction.do;jsessionid=4BA94EDED8F047B4BDCB649864D03E0E?Id=4>)
  + <http://www.devmedia.com.br/diferencas-entre-string-stringbuilder-e-stringbuffer-em-java/29865>

**private** **static** **void** strings() {

String texto0 = "texto opcional";

texto0 = texto0 + "linha 1";

texto0 = texto0 + "linha 2";

texto0 = texto0 + "linha 3";

System.***out***.println(texto0);

StringBuilder texto1 = **new** StringBuilder("texto opcional");

texto1.append("linha 1");

texto1.append("linha 2");

texto1.append("linha 3");

System.***out***.println(texto1);

StringBuffer texto2 = **new** StringBuffer("texto opcional");

texto2.append("linha 1");

texto2.append("linha 2");

texto2.append("linha 3");

System.***out***.println(texto2);

}

* Replace
* Substring
* Split

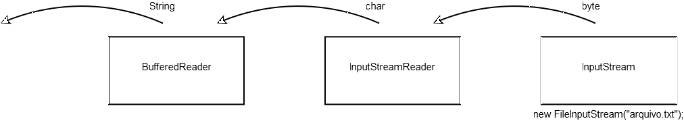
String texto4 = "a,w,e,r,r,t,t,y,d,f,a,e";

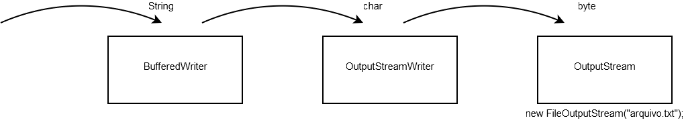
List textos = Arrays.*asList*(texto4.split(","));

System.***out***.println(textos.size());

**Java IO**

* InputStream e OutputStream





* Java 6 usando close e Java 7 não
  + <http://www.javapractices.com/topic/TopicAction.do;jsessionid=4BA94EDED8F047B4BDCB649864D03E0E?Id=57>
  + dica: sempre usar BufferedReader e BufferedWriter

**private** **static** **void** inputOutputJava6() {

List<String> comandos = Arrays.*asList*("cima", "baixo", "direita", "esquerda");

*serializarObjeto*(comandos);

*deserializar*();

}

**private** **static** **void** deserializar() {

**try** {

// use buffering

InputStream file = **new** FileInputStream("comandos.txt");

InputStream buffer = **new** BufferedInputStream(file);

ObjectInput input = **new** ObjectInputStream(buffer);

**try** {

List<String> lista = (List<String>) input.readObject();

**for** (String comando : lista) {

System.***out***.println("Comando: " + comando);

}

} **finally** {

input.close();

}

} **catch** (ClassNotFoundException ex) {

System.***out***.println("Erro: " + ex);

} **catch** (IOException ex) {

System.***out***.println("Erro: " + ex);

}

}

**private** **static** **void** serializarObjeto(List<String> comandos) {

**try** {

OutputStream file = **new** FileOutputStream("comandos.txt");

OutputStream buffer = **new** BufferedOutputStream(file);

ObjectOutput output = **new** ObjectOutputStream(buffer);

**try** {

output.writeObject(comandos);

} **finally** {

output.close();

}

} **catch** (IOException ex) {

System.***out***.println("Erro: " + ex);

}

}

**private** **static** **void** inputOutputJava7() {

**try** {

InputStreamReader reader = **new** FileReader("comandos.txt");

BufferedReader bufferReader = **new** BufferedReader(reader);

**try** (BufferedWriter bw = **new** BufferedWriter(**new** FileWriter("comandosa1.txt"))) {

String linha = bufferReader.readLine();

**while** (linha != **null**) {

System.***out***.println("Comando: " + linha);

bw.write(linha);

bw.newLine();

linha = bufferReader.readLine();

}

} **finally** {

reader.close();

}

} **catch** (IOException ex) {

System.***out***.println("Erro: " + ex);

}

}

* Java 7
  + Path, Scanner e PrintStream
    - <http://www.javapractices.com/topic/TopicAction.do;jsessionid=4BA94EDED8F047B4BDCB649864D03E0E?Id=42>

**private** **static** **void** inputOutputJava7() {

Path path = Paths.*get*("teste.txt");

**try** {

PrintStream novoArquivo = **new** PrintStream("saida.txt");

**try** (Scanner scanner = **new** Scanner(path)) {

**while** (scanner.hasNextLine()) {

String linha = scanner.nextLine();

System.***out***.println(linha);

novoArquivo.println(linha);

}

} **catch** (IOException e) {

e.printStackTrace();

}

} **catch** (FileNotFoundException e1) {

e1.printStackTrace();

}

}

* Copiar arquivos
  + <http://www.javapractices.com/topic/TopicAction.do;jsessionid=4BA94EDED8F047B4BDCB649864D03E0E?Id=246>

**private** **static** **void** copiarArquivo() {

Path FROM = Paths.*get*("saida.txt");

Path TO = Paths.*get*("e:\\temp\\saida2.txt");

CopyOption[] options = **new** CopyOption[] { StandardCopyOption.***REPLACE\_EXISTING***,

StandardCopyOption.***COPY\_ATTRIBUTES*** };

**try** {

Files.*copy*(FROM, TO, options);

} **catch** (IOException e) {

e.printStackTrace();

}

}

**Thread**

* Implements Runnable
* Threads.start

**public** **class** EscreveAlgo **implements** Runnable {

**public** **void** run() {

Integer contador = 0;

**while** (contador < 10000){

System.***out***.println("OUTRA COISA");

contador++;

}

}

}

**public** **class** EscreveOutraCoisa **implements** Runnable{

**public** **void** run() {

Integer contador = 0;

**while** (contador < 10000){

System.***out***.println("ALGO");

contador++;

}

}

}

**private** **static** **void** threads() {

EscreveAlgo algo = **new** EscreveAlgo();

EscreveOutraCoisa outraCoisa = **new** EscreveOutraCoisa();

Thread thread1 = **new** Thread(algo);

Thread thread2 = **new** Thread(outraCoisa);

thread1.start();

thread2.start();

}

* Syncronized

**public** **class** ClasseSynchronized **implements** Runnable {

**private** Integer valor = 200;

**public** **void** run() {

somaValorCem();

}

**private** **void** somaValorCem() {

**synchronized** (**this**) {

valor = valor + 100;

System.***out***.println("valor:" + valor);

}

}

}

**private** **static** **void** threadsSincronizadas() {

ClasseSynchronized sincronismo = **new** ClasseSynchronized();

Thread t1 = **new** Thread(sincronismo);

Thread t2 = **new** Thread(sincronismo);

t1.start();

t2.start();

}

**Coleções (Java 7 e Java 8)**

* List (permite valores semelhantes)
  + ArrayList
  + Sort (ordenar) + implementação Comparable<T>

**public** **int** compareTo(Classe outraClasse) {

**if** (**this**.atributo < outraClasse.atributo) {

**return** -1;

}

**if** (**this**.saldo > outra.saldo) {

**return** 1;

}

**return** 0;

}

* + LinkedList
    - Performance boa para add e remove
* Set (não se repete)
  + HashSet
  + LinkedHashSetList
    - Performance boa para add e remove
* Map(chave valor)
  + HashMap

Map<String, Classe> mapa = **new** HashMap<String, Classe>();

// adiciona duas chaves e seus respectivos valores

mapaDeContas.put("texto1", classe1);

mapaDeContas.put("texto2", classe2);

* Arrays
  + Arrays.asList(valor)
* Performance

**private** **static** **void** performanceHashSet() {

System.***out***.println("Iniciando Hashset...");

Collection<Integer> teste = **new** HashSet<Integer>();

*percorrer*(teste);

}

**private** **static** **void** performanceLinkedSet() {

System.***out***.println("Iniciando LinkedHashSet...");

Collection<Integer> teste = **new** LinkedHashSet<Integer>();

*percorrer*(teste);

}

**private** **static** **void** performanceList() {

System.***out***.println("Iniciando List...");

Collection<Integer> teste = **new** ArrayList<Integer>();

*percorrer*(teste);

}

**private** **static** **void** performanceLinkedList() {

System.***out***.println("Iniciando LinkedList...");

Collection<Integer> teste = **new** LinkedList<Integer>();

*percorrer*(teste);

}

**private** **static** **void** percorrer(Collection<Integer> teste) {

**long** inicio = System.*currentTimeMillis*();

**int** total = 30000;

**for** (**int** i = 0; i < total; i++) {

teste.add(i);

}

**for** (**int** i = 0; i < total; i++) {

teste.contains(i);

}

**long** fim = System.*currentTimeMillis*();

**long** tempo = fim - inicio;

System.***out***.println("Tempo gasto: " + tempo);

}

https://dzone.com/articles/java-collection-performance

* Comparable e comparator
  + <http://blog.caelum.com.br/ordenando-colecoes-com-comparable-e-comparator/>

**private** **static** **void** ordernarString() {

List<String> palavras = Arrays.*asList*("Fusca", "Wagner", "Ariane", "Mariah");

Collections.*sort*(palavras);

System.***out***.println(palavras);

}

**private** **static** **void** ordenarPorTamanhoString() {

List<String> palavras = Arrays.*asList*("Fuscaaaaaaaaaaaaaaa", "Wagnaaaaaaaaer", "Ariane", "Mariah");

Comparator<String> comparadorPorTamanho = **new** ComparadorPorTamanho();

Collections.*sort*(palavras, comparadorPorTamanho);

System.***out***.println(palavras);

}

**class** ComparadorPorTamanho **implements** Comparator<String>{

@Override

**public** **int** compare(String o1, String o2) {

**if** (o1.length() < o2.length()){

**return** -1;

}

**if** (o1.length() > o2.length()){

**return** 1;

}

**return** 0;

}

}

**Java 8**

* Coleção com ordenação
  + palavras.sort(comparadorPorTamanho);
  + **Métodos default**
    - Métodos que tem corpo dentro de uma interface
* Foreach

**Já existente**

**for** (String p:palavras){

System.***out***.println(p);

}

Novo

palavras.forEach(new ImprimeTexto());

**class** ImprimeTexto **implements** Consumer<String>{

@Override

**public** **void** accept(String t) {

System.***out***.println(t);

}

}

* Lambda
  + Código Atual

palavras.forEach(new ImprimeTexto());

* + Novo Código

palavras.forEach((String s) -> {

System.out.println(s);

});

* + Melhorando...

palavras.forEach(s -> System.out.println(s));

* + Interface Funcional
    - Interface com 1 metodo abstrato

palavras.sort((String o1, String o2) -> {

if (o1.length() < o2.length()) {

return -1;

}

if (o1.length() > o2.length()) {

return 1;

}

return 0;

});

* + **Melhorando**...

palavras.sort((String o1, String o2) -> {

return Integer.compare(o1.length(), o2.length());

});

* + **Melhorando**

palavras.sort((String o1, String o2) -> Integer.compare(o1.length(), o2.length()));

* Method reference
  + Maneira de escrever lambda mais enxuto
  + Só para lambdas bem enxutos
    - palavras.sort(Comparator.comparing(s -> s.length()));
    - palavras.sort(Comparator.comparing(String :: length));
* stream
  + Toda coleção possui um stream (uma corrente de objetos)
  + O que é feito na stream não gera impacto na coleção original
  + Permite muitos opcionais
    - Filter
      * https://www.mkyong.com/java8/java-8-streams-filter-examples/
    - Foreach
    - sorted
      * [https://www.mkyong.com/java8/java-8-lambda-comparator-example /](https://www.mkyong.com/java8/java-8-lambda-comparator-example%09/)
    - findAny (qualquer um)

Optional<Pessoa> opt = pessoas.stream().filter(p -> p.getIdade() >=32).findAny();

Pessoa pessoa = opt.orElse(**null**);

System.***out***.println(pessoa);

* + - FindFirst (só o primeiro)
    - IfPresent
      * opt.ifPresent(p->System.***out***.println(p.getNome()));
      * ou
      * Optional<Pessoa> opt = pessoas.stream().filter(p -> p.getIdade() >=32).findAny().ifPresent(p->System.***out***.println(p.getNome()));
    - Collectors
      * List<Pessoa> filtrado = pessoas.stream().filter(p -> p.getIdade() >=32).collect(Collectors.*toList*());
    - Optional
      * Permite trabalhar com objetos que tem nulo

**public** **class** Pessoa {

**private** String nome;

**private** Integer idade;

**public** Pessoa(String nome, Integer idade) {

**this**.nome = nome;

**this**.idade = idade;

}

**public** String getNome() {

**return** nome;

}

**public** **void** setNome(String nome) {

**this**.nome = nome;

}

**public** Integer getIdade() {

**return** idade;

}

**public** **void** setIdade(Integer idade) {

**this**.idade = idade;

}

@Override

**public** String toString() {

**return** "Pessoa [nome=" + nome + "]";

}

}

**private** **static** **void** stream() {

List<Pessoa> pessoas = **new** ArrayList<>();

pessoas.add(**new** Pessoa("maria", 32));

pessoas.add(**new** Pessoa("jose", 31));

pessoas.add(**new** Pessoa("jesus", 1));

pessoas.add(**new** Pessoa("pilatos", 36));

pessoas.sort(Comparator.*comparing*(Pessoa::getIdade));

//pessoas.forEach(p -> System.out.println(p.getNome()));

pessoas.stream().filter(p -> p.getIdade() >=32).forEach(p -> System.***out***.println(p.getNome()));

}

* Datas
  + Atuais DATE e Calendar
  + Substitui JodaTime

**private** **static** **void** apiData() {

LocalDate hoje = LocalDate.*now*();

System.***out***.println(hoje);

LocalDate copa2018 = LocalDate.*of*(2018, Month.***JUNE***, 03);

**int** anos = copa2018.getYear() - hoje.getYear();

System.***out***.println(anos);

Period periodo = Period.*between*(hoje, copa2018);

System.***out***.println(periodo.getYears());

LocalDate novaData = copa2018.plusMonths(4);

DateTimeFormatter formatador = DateTimeFormatter.*ofPattern*("dd/MM/yyyy");

System.***out***.println(copa2018.format(formatador));

}

* Data e Hora
  + Se precisar de horas, usar o LocalDateTime

**links**:

* <https://javaantipatterns.wordpress.com/>
* <http://www.javapractices.com/home/HomeAction.do>

**Exercícios**

Todos os exercícios devem ter testes unitários para eles.

1. Faça um método que receba um texto com acentuação e devolva alguns caracteres sem acentuação.
   * Caso passe ã, transformar em a
   * Caso passe á, transformar em a
   * Caso passe í, transformar em i
   * Caso passe ç, transformar em c
2. Faça um método que receba o texto “AULA DE SÁBADO” e que mude a palavra AULA para ÁULÁ
3. Faça um método que receba um texto e que seja exibida cada caracter deste texto
4. Faça um método que receba um texto com vírgulas e mostre-o separado pelas virgulas.