RICARDO LEONARDO PANSONATO 29SCJ

1. Dado o programa abaixo, responda aos questionamentos a seguir, ao final da execução:

**import** java.lang.reflect.Field;

**import** sun.misc.Unsafe;

**public** **class** Exercicio01 {

**static** **final** Unsafe ***unsafe*** = *getUnsafe*();

**static** **final** **boolean** ***is64bit*** = **true**;

**public** **static** **void** printAddresses(String label, Object... objects) {

System.***out***.print(label + ": 0x");

**long** last = 0;

**long** offset = ***unsafe***.arrayBaseOffset(objects.getClass());

**int** scale = ***unsafe***.arrayIndexScale(objects.getClass());

**switch** (scale) {

**case** 4:

**long** factor = ***is64bit*** ? 8 : 1;

**final** **long** i1 = (***unsafe***.getInt(objects, offset) & 0xFFFFFFFFL) \* factor;

System.***out***.print(Long.*toHexString*(i1));

last = i1;

**for** (**int** i = 1; i < objects.length; i++) {

**final** **long** i2 = (***unsafe***.getInt(objects, offset + i \* 4) & 0xFFFFFFFFL) \* factor;

**if** (i2 > last)

System.***out***.print(", +" + Long.*toHexString*(i2 - last));

**else**

System.***out***.print(", -" + Long.*toHexString*( last - i2));

last = i2;

}

**break**;

**case** 8:

**throw** **new** AssertionError("Not supported");

}

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

}

**private** **static** Unsafe getUnsafe() {

**try** {

Field theUnsafe = Unsafe.**class**.getDeclaredField("theUnsafe");

theUnsafe.setAccessible(**true**);

**return** (Unsafe) theUnsafe.get(**null**);

} **catch** (Exception e) {

**throw** **new** AssertionError(e);

}

}

**public** **static** **void** main(String[] args) {

String string = "";

**int** z = 1;

*printAddresses*(z + ": string", string);

**for** (**int** i = 0; i < 12; i++) {

string += "string";

z++;

*printAddresses*(z + ": string", string);

}

z = 1;

StringBuilder sb = **new** StringBuilder("");

*printAddresses*(z + ": sb", sb);

**for** (**int** i = 0; i < 12; i++) {

sb.append("teste");

z++;

*printAddresses*(z + ": sb", sb);

}

z = 1;

StringBuffer sbf = **new** StringBuffer("");

*printAddresses*(z + ": sbf", sbf);

**for** (**int** i = 0; i < 12; i++) {

sbf.append("teste");

z++;

*printAddresses*(z + ": sbf", sbf);

}

}

}

* 1. Foram criados 13 objetos
  2. Strings são imutáveis e são armazenadas em um pool. Para cada concatenação, um novo objeto é criado, logo, se alterarmos para StringBuffer (controle de concorrência) ou StringBuilder (sem controle de concorrência), trabalharemos com uma instância desse objeto, já que ele foi criado fora do laço, e depois invocamos o append apenas para as novas partes da String, sem ter de copiar o que já foi previamente processado.

1. Desenvolva um sistema que imprima em tela a data e hora atual nos seguintes formatos, considerando a data 10/09/2012 12:00:
   1. 2012-09-10 12:00
   2. 10-09-12 12:00:00
   3. Monday (dia da semana)
   4. 2012 (ano)
   5. AD (era)

Utilize apenas formatadores baseado no DateTimeFormatter para cada tipo de formato.

**import** java.time.LocalDateTime;

**import** java.time.format.DateTimeFormatter;

**import** java.util.Locale;

**public** **class** Exercicio02 {

**public** **static** **void** main(String[] args) {

DateTimeFormatter formatDate = DateTimeFormatter.*ofPattern*("dd/MM/yyyy HH:mm");

LocalDateTime date = LocalDateTime.*parse*("10/09/2012 12:00", formatDate);

System.***out***.println(DateTimeFormatter.*ofPattern*("yyyy-MM-dd HH:mm").format(date));

System.***out***.println(DateTimeFormatter.*ofPattern*("dd-MM-yy HH:mm:ss").format(date));

System.***out***.println(DateTimeFormatter.*ofPattern*("EEEE", Locale.***ENGLISH***).format(date));

System.***out***.println(DateTimeFormatter.*ofPattern*("yyyy").format(date));

System.***out***.println(DateTimeFormatter.*ofPattern*("G", Locale.***ENGLISH***).format(date));

}

}

1. DateTimeFormatter.*ofPattern*("yyyy-MM-dd HH:mm").format(date)
2. DateTimeFormatter.*ofPattern*("dd-MM-yy HH:mm:ss").format(date)
3. DateTimeFormatter.*ofPattern*("EEEE", Locale.***ENGLISH***).format(date)
4. DateTimeFormatter.*ofPattern*("yyyy").format(date)
5. DateTimeFormatter.*ofPattern*("G", Locale.***ENGLISH***).format(date)
6. Crie um sistema que seja responsável por medir a eficiência de busca de alguns tipos de objetos da classe de coleções do Java. O sistema deverá incluir nos determinados tipos de coleções 1.000.000 entradas do tipo inteiro de números sequenciais. Utilize um laço para efetuar a inserção dos valores. Depois dos dados estarem nas coleções, deverá ser solicitado a busca da entrada 500.000. Para coleções que necessitam de uma chave, utilize o mesmo valor da entrada, por exemplo, para o valor de entrada 1, a chave também será 1. Compare o tempo tanto de inserção de dados quanto de busca para cada tipo de coleção que será solicitada abaixo. Para a comparação de tempo, utilize sempre na unidade de milissegundos. Por meio deste aplicativo, indique quais são os melhores tipos de coleções para inserção e busca de informação.
   1. ArrayList e LinkedList
   2. Stack e HashSet
   3. HashMap e WeakHashMap

**import** java.time.LocalDateTime;

**import** java.time.temporal.ChronoUnit;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.HashSet;

**import** java.util.LinkedList;

**import** java.util.List;

**import** java.util.Stack;

**import** java.util.WeakHashMap;

**public** **class** Exercicio03 {

**private** **static** **final** **int** ***MAX\_CAPACITY*** = 1\_000\_000;

**private** **static** **final** **int** ***SEARCH\_POSITION*** = 500\_000;

**public** **static** **void** main(String[] args) {

**long** diff1 = 0l, diff2 = 0l;

LocalDateTime dateTime = LocalDateTime.*now*();

List<Integer> arrayList = **new** ArrayList<>(***MAX\_CAPACITY***);

**for** (**int** i = 0; i < ***MAX\_CAPACITY***; i++) {

arrayList.add(i);

}

diff1 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

dateTime = LocalDateTime.*now*();

List<Integer> linkedList = **new** LinkedList<>();

**for** (**int** i = 0; i < ***MAX\_CAPACITY***; i++) {

linkedList.add(i);

}

diff2 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

**if** (diff1 < diff2) {

System.***out***.println("ArrayList é melhor para inserção");

} **else** **if** (diff1 == diff2) {

System.***out***.println("ArrayList e LinkedList possuem o mesmo tempo para inserção");

} **else** {

System.***out***.println("LinkedList é melhor para inserção");

}

dateTime = LocalDateTime.*now*();

Stack<Integer> stack = **new** Stack<>();

**for** (**int** i = 0; i < ***MAX\_CAPACITY***; i++) {

stack.add(i);

}

diff1 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

dateTime = LocalDateTime.*now*();

HashSet<Integer> hashSet = **new** HashSet<>();

**for** (**int** i = 0; i < ***MAX\_CAPACITY***; i++) {

hashSet.add(i);

}

diff2 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

**if** (diff1 < diff2) {

System.***out***.println("Stack é melhor para inserção");

} **else** **if** (diff1 == diff2) {

System.***out***.println("Stack e HashSet possuem o mesmo tempo para inserção");

} **else** {

System.***out***.println("HashSet é melhor para inserção");

}

dateTime = LocalDateTime.*now*();

HashMap<Integer, String> hashMap = **new** HashMap<>();

**for** (**int** i = 0; i < ***MAX\_CAPACITY***; i++) {

hashMap.put(i, "teste" + i);

}

diff1 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

dateTime = LocalDateTime.*now*();

WeakHashMap<Integer, String> weakHashMap = **new** WeakHashMap<>();

**for** (**int** i = 0; i < ***MAX\_CAPACITY***; i++) {

weakHashMap.put(i, "teste" + i);

}

diff2 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

**if** (diff1 < diff2) {

System.***out***.println("HashMap é melhor para inserção");

} **else** **if** (diff1 == diff2) {

System.***out***.println("HashMap e WeakHashMap possuem o mesmo tempo para inserção");

} **else** {

System.***out***.println("WeakHashMap é melhor para inserção");

}

dateTime = LocalDateTime.*now*();

arrayList.get(***SEARCH\_POSITION***);

diff1 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

dateTime = LocalDateTime.*now*();

linkedList.get(***SEARCH\_POSITION***);

diff2 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

**if** (diff1 < diff2) {

System.***out***.println("ArrayList é melhor para pesquisa");

} **else** **if** (diff1 == diff2) {

System.***out***.println("ArrayList e LinkedList possuem o mesmo tempo para pesquisa");

} **else** {

System.***out***.println("LinkedList é melhor para inserção");

}

dateTime = LocalDateTime.*now*();

stack.get(***SEARCH\_POSITION***);

diff1 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

dateTime = LocalDateTime.*now*();

hashSet.contains(***SEARCH\_POSITION***);

diff2 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

**if** (diff1 < diff2) {

System.***out***.println("Stack é melhor para pesquisa");

} **else** **if** (diff1 == diff2) {

System.***out***.println("Stack e HashSet possuem o mesmo tempo para pesquisa");

} **else** {

System.***out***.println("HashSet é melhor para inserção");

}

dateTime = LocalDateTime.*now*();

hashMap.get(***SEARCH\_POSITION***);

diff1 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

dateTime = LocalDateTime.*now*();

weakHashMap.get(***SEARCH\_POSITION***);

diff2 = ChronoUnit.***MILLIS***.between(dateTime, LocalDateTime.*now*());

**if** (diff1 < diff2) {

System.***out***.println("HashMap é melhor para pesquisa");

} **else** **if** (diff1 == diff2) {

System.***out***.println("HashMap e WeakHashMap possuem o mesmo tempo para pesquisa");

} **else** {

System.***out***.println("WeakHashMap é melhor para inserção");

}

}

}

**OUTPUT:**

a) ArrayList é melhor para inserção

b) Stack é melhor para inserção

c) WeakHashMap é melhor para inserção

a) ArrayList é melhor para pesquisa

b) Stack e HashSet possuem o mesmo tempo para pesquisa

c) HashMap e WeakHashMap possuem o mesmo tempo para pesquisa

1. Desenvolva um aplicativo onde deverá ser criada uma classe chamada Pessoa com os seguintes atributos: nome (String), idade (int), sexo (char) e empresa (String). Depois crie uma coleção do tipo ArrayList de 5 objetos desta classe conforme tabela abaixo. Mostre como resultado em console, as seguintes informações solicitadas, utilizando a classe utilitária Collections (pode ser necessário alguma implementação na classe Pessoa para a execução de determinados itens):
   1. Sortear a lista e retirar uma pessoa;
   2. Ordenação da lista baseado no atributo nome;
   3. Inverter a ordem da lista baseado no atributo nome;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.List;

**public** **class** Exercicio04 {

**public** **static** **void** main(String[] args) {

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

pessoas.add(**new** Pessoa("João Garcia", 20, 'M', "AIK Enterprises"));

pessoas.add(**new** Pessoa("Maria Martins", 44, 'F', "Simples"));

pessoas.add(**new** Pessoa("Henrique Fernando", 43, 'M', "AIK Enterprises"));

pessoas.add(**new** Pessoa("Inácio Luís", 33, 'M', "Magazine André"));

pessoas.add(**new** Pessoa("Fernando Ferreira", 87, 'M', "Casas Recife"));

Collections.*shuffle*(pessoas);

System.***out***.println("a) " + pessoas.get(0));

Collections.*sort*(pessoas, (u1, u2) -> u1.getNome().compareTo(u2.getNome()));

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

pessoas.forEach(s -> {

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

});

Collections.*reverse*(pessoas);

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

pessoas.forEach(s -> {

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

});

}

}

OUTPUT:

a) Pessoa [nome=Inácio Luís, idade=33, sexo=M, empresa=Magazine André]

b)

Pessoa [nome=Fernando Ferreira, idade=87, sexo=M, empresa=Casas Recife]

Pessoa [nome=Henrique Fernando, idade=43, sexo=M, empresa=AIK Enterprises]

Pessoa [nome=Inácio Luís, idade=33, sexo=M, empresa=Magazine André]

Pessoa [nome=João Garcia, idade=20, sexo=M, empresa=AIK Enterprises]

Pessoa [nome=Maria Martins, idade=44, sexo=F, empresa=Simples]

c)

Pessoa [nome=Maria Martins, idade=44, sexo=F, empresa=Simples]

Pessoa [nome=João Garcia, idade=20, sexo=M, empresa=AIK Enterprises]

Pessoa [nome=Inácio Luís, idade=33, sexo=M, empresa=Magazine André]

Pessoa [nome=Henrique Fernando, idade=43, sexo=M, empresa=AIK Enterprises]

Pessoa [nome=Fernando Ferreira, idade=87, sexo=M, empresa=Casas Recife]

1. Desenvolva um sistema baseado em criptografia que tem o objetivo de codificar e decodificar um arquivo texto. A regra de codificação é incrementar o código ASCII de cada caractere, de forma que o código 1, representado pela letra A seria alterado para 2 (1+2), representado pela letra B e assim por diante. O sistema deve funcionar da seguinte forma: 1) solicitar uma mensagem ao usuário a ser codificada; 2) codificar e salvar o arquivo “mensagemCodificada.txt” e por fim 3) salvar o arquivo decodificado no arquivo “mesagemDecodificada.txt”. Utilize o StringBuilder para alterar os valores caractere a caractere, utilizando o método setChar(int, char).

**import** java.io.FileReader;

**import** java.io.PrintWriter;

**import** java.util.Scanner;

**public** **class** Exercicio05 {

**public** **static** **void** main(String[] args) {

StringBuilder sb = **new** StringBuilder();

**try** (Scanner s = **new** Scanner(System.***in***)) {

System.***out***.println("Mensagem: ");

String mensagem = s.nextLine();

mensagem.chars().forEach(c -> {

**char** nextChar = (**char**) (c + 1);

sb.append(nextChar);

});

}

**try** (PrintWriter writer = **new** PrintWriter("mensagemCodificada.txt", "UTF-8")) {

System.***out***.println("Gravando a mensagem codificada " + sb.toString());

writer.println(sb.toString());

} **catch** (**final** Exception e) {

System.***out***.println("Não foi possível criar o arquivo mensagemCodificada.txt");

}

**try** (FileReader reader = **new** FileReader("mensagemCodificada.txt");

Scanner in = **new** Scanner(reader);

PrintWriter writer = **new** PrintWriter("mensagemDecodificada.txt", "UTF-8")) {

String mensagemCodificada = in.nextLine();

StringBuilder sbDecoficado = **new** StringBuilder();

mensagemCodificada.chars().forEach(c -> {

**char** nextChar = (**char**) (c - 1);

sbDecoficado.append(nextChar);

});

System.***out***.println("Gravando a mensagem decodificada " + sbDecoficado.toString());

writer.println(sbDecoficado.toString());

} **catch** (**final** Exception e) {

System.***out***.println("Não foi possível criar o arquivo mensagemDecodificada.txt");

}

}

}