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