Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Linguagens e Ambientes de Execução

Exercícios #2, Verão de 2022/2023

Dificuldade: ●●●●○

1. [3] Considere o programa Kotlin do ficheiro fonte indicado abaixo.

classes.kt

```
import java.io.Closeable

interface Printable { fun printIt() }

class Info(val code : Int, val text : String) : Printable, Closeable {
        override fun printIt() { println("Info($code) { text: \"$text\"}") }
        override fun close()   { println("Info($code) -- Done") }
}

fun texts() = sequence { yield("ISEL"); yield("LEIC"); yield("LAE") }

fun main() {
        var count = 1
        texts().map { Info(count++, it) }.forEach { it.use { info ->    info.printIt() } }
}
```

   a. [1] Diga, **justificando**, quantos ficheiros `.class` resultam da sua compilação e indique os respectivos nomes, quando não contêm o carácter $.

   b. [1] Indique, **justificando**, qual a instrução de *bytecode* Java usada para ler o valor da variável `count` em `Info(count++, it)` na função `main`.

   c. [1] Indique, **justificando**, qual o número de estados da máquina de estados (*invokeSuspend*) gerada para a sequência da função `texts`.

2. [5] Pretende-se desenvolver, em Java, o método estático genérico `VarUtils.getPublicVarGettersOf` que retorna um mapa a associar os nomes das propriedades publicamente mutáveis do tipo `T`, excepto as anotadas com `@get:DontRead`, às instâncias de `Method` correspondentes aos respectivos métodos *getter*.

   a. [3] Escreva em Kotlin a anotação `@DontRead`, aplicável apenas a *getters*, e apresente em Java o método estático `getPublicVarGettersOf`, incluindo a sua assinatura completa.

   b. [2] Complete o método `showVars`, tirando partido do método `getPublicVarGettersOf`, de modo a que o programa de exemplo (abaixo, à esquerda) produza no *standard output* o texto (não sublinhado) que se vê à direita do exemplo.

```
class Foo(a: Int, var b: String, val c: Int, d: Double) {
    @get:DontRead var n = "HIDDEN"
    val x : String = "ISEL"
    var y : Int = 2023
    private var z : Double = d * 2
}

fun main() {
    val obj = Foo(1000, "LEIC", 888, 3.14159)
    showVars(obj)
}
```

OUTPUT

```
Foo {
    var b : String = LEIC
    var y : int = 2023
}
```

```
public class VarUtils {
    public static void showVars(Object obj) { … }
}
```

3. [1.5] Ordene as classes seguintes pelo tamanho do espaço que as suas instâncias ocupam no *heap*, justificando:

```
class X(bar: Int) {
    val nr: Int
        get() = foo
    companion object {
        val foo = 7657
        val zaz = 1234
        val qux = 9876
    }
}
```

```
class Y(var bar: Int = 7657,
        qux: String = "ISEL-LEIC") {
    val nr
        get() = bar
    val foo
        get() = nr
}
```

```
class Z(
    val bar: Int,
    val foo: Int
)
```

4. [3.5] O troço de código à esquerda, suportado pela biblioteca Cojen/Maker, é usado para gerar o código fonte de um tipo que será posteriormente utilizado num programa Kotlin. O excerto de *bytecode* à direita resulta da compilação de um método de extensão, escrito em Kotlin, para o tipo gerado pelo código à esquerda (estão omitidos os dois `checkNotNullParameter` que são realizados sobre os parâmetros do método de extensão).

```
fun generateCode() : ClassMaker {
    val cm = ClassMaker.beginExternal("Valuer")
                       .interface_().public_()
    cm.addMethod(Int::class.java, "estimate",
                 String::class.java)
                 .public_().abstract_()
    return cm
}
```

```
12: aload_0
13: aload_1
14: invokeinterface #23,  2
19: iconst_2
20: iadd
21: iconst_5
22: idiv
23: ireturn
```

   a. [1] Apresente código fonte Java ou Kotlin equivalente ao gerado pelo troço de código à esquerda.

   b. [1.5] Apresente o código fonte Kotlin que dá origem ao *bytecode* apresentado à direita.

   c. [1] Para cada uma das frases abaixo, indique se é verdadeira ou falsa, com a devida justificação:

      i. Se um programa Kotlin incluir o ficheiro `.class` com o *bytecode* indicado à direita, então o gerador de código apresentado à esquerda terá de ser executado, o mais tardar, logo antes da primeira execução do método de extensão.

      ii. Considere dois ficheiros: 1. O ficheiro `.class` produzido pelo gerador da esquerda; e 2. O ficheiro com o código fonte Kotlin do método de extensão. Bastam esses dois ficheiros, para que o compilador de Kotlin possa produzir o *bytecode* apresentado à direita.

      iii. Se um programa compilado para a JVM incluir o ficheiro `.class` com o *bytecode* indicado à direita, então o ficheiro `.class` produzido pelo gerador da esquerda não é necessário para a sua execução.

      iv. Se um programa compilado para a JVM incluir o ficheiro `.class` com o *bytecode* indicado à direita, então é inútil que o mesmo programa também inclua o gerador apresentado à esquerda.

5. [1] Considere a variável `strs`, do tipo `MutableList<String>`:

```
val strs = mutableListOf("ISEL", "LEIC", "LAE")
```

   Como se consegue colocar um valor de tipo `Int` nesta lista?

   i. `strs[2] as Int = 5`

   ii. `(strs as Array<Int>)[2] = 5`

   iii. `(strs as MutableList<Int>).add(5)`

   iv. `strs.add(5 as String)`

6. [2] Identifique na listagem da função `hot()` as instruções que podem gerar operações de *boxing*, *unboxing* e *checkcast*, justificando.

```
fun hot() {
   val other: Int?
   val n = 6545
   other = n
   val res = n.equals("ola")
   val end = other + res as Int
}
```

7. [2] Qual o *output* da execução do seguinte programa?
   Se no lugar de `sequenceOf` usasse `arrayOf` existiria alguma diferença? Justifique.

```
val nrs = sequenceOf("abc", "def", "super").map { print("$it ");  it.length }
println(nrs.distinct().count())
println(nrs.count())
```

8. [1] Um dos *garbage collectors* disponível na JVM incluída no JDK17 é o G1, que a Oracle sumariza assim: «*G1 is a generational, incremental, parallel, mostly concurrent, stop-the-world, and evacuating garbage collector*». O que significa, no âmbito de um *garbage collector*, cada um dos termos sublinhados?

9. [1] A utilização de uma instância da classe OutFile corretamente construída (o construtor correu bem) pode falhar na execução de uma das suas instruções. Identifique qual a instrução que pode falhar e justifique em que situação.
   ATENÇÃO: admita que **não** existem interferências do *file system* que provoquem esse erro. A origem da falha está na forma de implementação da classe OutFile.

```
class OutFile(path: String) : Closeable {
   private var out : OutputStream? = FileOutputStream(path)
   fun write(msg: String) = out?.write(msg.toByteArray())
   override fun close() = cleanup()
   protected fun finalize() = cleanup()
   private fun cleanup() { out!!.close(); out = null }
}
```

Duração: ilimitada
ISEL, 31 de maio de 2023