Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Linguagens e Ambientes de Execução

Época de Recurso, Verão de 2021/2022, 15 de Julho de 2022

Grupo 1

1. [2] Considere o código fonte Kotlin indicado abaixo, presente no ficheiro *code.kt*. Indique os nomes de todos os ficheiros `.class` que resultam da sua compilação e em qual/quais deles ficam as definições nativas de **`e`** e **`pi`**.

```
val e  = 2.718281828
val pi = 3.141592654

interface F  { fun op(arg : Double) : Double }

object Stats { var maxOffset = 0.0 }

class A : F  { override fun op(arg : Double) : Double { /* ... */ } }

class B(private val offset : Double) : F {
   init { Stats.maxOffset = max(Stats.maxOffset, offset) }
   override fun op(arg : Double) : Double { /* ... */ }
}
```

2. [2] Ordene as classes seguintes pelo tamanho do espaço que as suas instâncias ocupam no *heap*, justificando:

```
class A() {
  fun foo() = 123456
  val nr
    get() = 7658894
  val foo
    get() = nr
}
```

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

```
class C(val bar: Int) {
  val nr: Int
    get() = bar
  fun foo() = nr
  fun oper(num: Int) : Int {
    val x = num + 1
    val y = num - 1
    return x * y
  }
}
```

3. [3] Considere o exemplo da classe `Account` cujas propriedades têm uma anotação `Check` que identifica a função que verifica se um valor é válido para aquela propriedade.

   Implemente a função **`fun checkAndSet(target: Any, values: Map<String, Any>)`** que atribui os valores do mapa **`values`** às propriedades de **`target`** correspondentes à chave do mapa, verificando que o valor é válido segundo a função anotada na propriedade. Caso não seja um valor válido, lança uma excepção.

   Assuma que as funções de verificação são métodos de instância de **`target`**.

```
class Account {
    @Check("nonNegative") var balance: Long,
    @Check("dotCom") var email: String

    fun nonNegative(v: Long) = v > 0
    fun dotCom(address: String) = address.endsWith(".com")
}
```

4. [2] Identifique na listagem da função yyy() as instruções que podem gerar operações de *boxing*, *unboxing* e *checkcast*, justificando.

```
1  fun win(obj: Any) = obj
2
3  fun yyy(): Any {
4    val n: Int = Random.nextInt()
5    val obj = win(n)
6    val p = obj as Int?
7    val res =  p!!
8    return res + n.hashCode()
9  }
```

(continua)

Grupo 2

5. [2] Apresente uma função em Kotlin equivalente à descrição em *bytecode* Java apresentada ao lado.

```
public static double boom(String, Object, int);
  Code:
     ALOAD 0
     INVOKEVIRTUAL java/lang/String.length ()I
     ALOAD 1
     INVOKEVIRTUAL java/lang/Object.hashCode ()I
     IDIV
     ILOAD 2
     ISUB
     IRETURN
```

6. [3] Considere o seguinte código Kotlin:

```
arrayOf("abc", "isel", "super")
     .map { print("$it ");  it.length }
     .filter { print("$it "); it == 4 }
     .first()
```

   a. [1.5] Qual o *output* da execução do código indicado?

   b. [1.5] Se no lugar de `arrayOf` usasse `sequenceOf`, existiria alguma diferença? Justifique.

7. [2] Implemente a função de extensão `List<Any>.castTo(): List<T>`, que retorna a mesma lista de entrada se todos os seus elementos forem compatíveis com T. Caso contrário lança uma excepção. Exemplo:

```
val objs1: List<Any> = listOf(1, 2, 3, 4, 5)
val objs2: List<Any> = listOf(LocalDate.of(2022, 7, 15), "2022-6-29")

val nbrs: List<Int> = objs1.castTo<Int>()              // OK
val strs: List<LocalDate> = objs2.castTo<LocalDate>() // throws Exception
```

8. [2] No contexto do *Garbage Collector* o que é uma *root reference* e qual a relevância no seu funcionamento?

9. [2] Na JVM correspondente à versão 9 do Java foi introduzido o mecanismo de *cleaners* para substituir o de *finalizers*. Para cada um dos casos seguintes, correspondentes a desvantagens do mecanismo de *finalizers*, indique de que forma o mecanismo de *cleaners* evita a desvantagem indicada. Justifique devidamente as suas respostas.

   a. [1] A definir o método `finalize` numa classe, todas as instâncias desse tipo são sujeitas ao processo de finalização, mesmo as que não precisam dele.

   b. [1] Mesmo quando o método `finalize` de uma classe apenas opera sobre um dos campos dessa classe, as instâncias completas da classe ficam a ocupar espaço em memória enquanto não são finalizadas.