Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Linguagens e Ambientes de Execução
Época Normal, Verão de 2021/2022, 29 de Junho de 2022

Grupo 1

1. [2] Considere o código fonte Kotlin indicado abaixo, presente no ficheiro *light.kt*. Indique os nomes dos ficheiros `.class` em que ficam as definições nativas de `Adjustable`, `Light`, `DIMMED` e `createLight`, após a compilação.

```
interface Adjustable { fun adjust(diff : Int) }

class Light(private var intensity : Int) : Adjustable {
   override fun adjust(diff : Int) { intensity += diff; update() }
   private fun update() { ... }
}

val DIMMED = 15

fun createLight(intensity: Int = DIMMED) = Light(intensity)
```

2. [2] Ordene as classes seguintes pelo tamanho do espaço que as suas instâncias ocupam no *heap*, justificando. Considere que é igual o espaço ocupado por Int ou Float.

```
class A (
  val x: Int,
  val y: Int
){
  fun modulus() : Float =
    sqrt(x.toFloat()*x + y*y)
}
```

```
class B (
  val x: Int,
  val y: Int,
  var modulus: Float = 0.0
)
```

```
class C (
  x: Int,
  y: Int,
  val modulus: Float =
    sqrt(x.toFloat()*x + y*y)
)
```

3. [3] Considere uma anotação que define a gama de valores aceites numa propriedade de tipo String. Por exemplo, se a propriedade **`desc`** na classe **`Weather`** só aceita os valores `"Sunny"`, `"Cloudy"`, `"Rainy"`, essa propriedade é anotada na forma:

```
@ValidText(arrayOf("Sunny", "Rainy", "Cloudy")) var desc: String
```

Implemente a anotação **`ValidText`** e a função **`checkAndSet`** que afecta uma propriedade se o valor recebido pertencer à gama de valores anotados na propriedade, caso contrário lança excepção. Exemplo:

```
checkAndSet(lisbonWeather, "desc", "Cloudy") // lisbonWeather.desc ⇦ "Cloudy"
checkAndSet(lisbonWeather, "desc", "Windy") // Lança IllegalArgumentException
```

4. [2] Apresente uma função em Kotlin equivalente à descrição em *bytecode* Java apresentada ao lado.

```
public static final double magn(float, float);
  Code:
    0: fload_0
    1: fload_0
    2: fmul
    3: fload_1
    4: fload_1
    5: fmul
    6: fadd
    7: f2d                  // (float to double)
    8: invokestatic  #23    // java/lang/Math.sqrt:(D)D
   11: dreturn
```

5. [2] Identifique na listagem da função **`bar`** as instruções e o tipo de operação de *boxing*, *unboxing* ou *checkcast* que pode ser gerado.

```
1  class Pack(val v: Any)
2
3  fun bar(): Int {
4    val n = 6235
5    val p = Pack(n)
6    val res =  p.v as Int?
7    return res ?: 0
8  }
```

Grupo 2

6. [3] Recorrendo a `sequence` e `yield`, construa a função de extensão `repeat`, que produz de forma *lazy* uma nova sequência que repete cada elemento da sequência de entrada o número de vezes especificado por parâmetro.
   Exemplo: `sequenceOf(3, 5, 1, 5).repeat(3)` produz uma sequência com 3, 3, 3, 5, 5, 5, 1, 1, 1, 5, 5, 5

```
fun <T : Any> Sequence<T>.repeat(times: Int): Sequence<T>
```

7. [2] Complete a função genérica `listOfDefaults<T>(n : Int)`, que retorna uma lista com `n` instâncias de `T`, em que cada instância é criada via `createInstance()`, usando o construtor sem parâmetros obrigatórios. Ignore a situação de erro quando o tipo `T` não tem um construtor compatível com `createInstance()`.

```
/* TO DO : completar assinatura */ listOfDefaults(n : Int = 0) : List<T> {
    val list = mutableListOf<T>()
    // TO DO: adicionar elementos com list.add
    return list
}

// Exemplo de utilização
fun main() {
    val list1 = listOfDefaults<Student>(3)
    val list2 : List<Person> = listOfDefaults(2)
    // ...
}
```

8. [2] O que se entende por *garbage collector* geracional e qual a ideia principal em que se fundamenta?

9. [2] Dada a definição de `FinishFile` indique e **justifique** o que é **escrito** no ficheiro *out.txt* como resultado da execução do `main` com cada uma das implementações de `func`.
   NOTE: `runFinalization()` só retorna quando tiverem sido executados todos os `finalize` pendentes.

```
class FinishFile(path: String) : Closeable{
  private val out = FileOutputStream(path)
  fun write(msg: String) =
    out.write(msg.toByteArray())
  fun close(msg: String) {
    write(msg)
    out.close()
  }
  override fun close() = close("CLOSED")
  protected fun finalize() = close("FINALIZED")
}
```

```
val titles =
  arrayOf("start", "begin", "init")

fun main() {
  try{
    func()
  } finally {
    System.gc()
    System.runFinalization()
  }
}
```

A

```
fun func() {
  val ff = FinishFile("out.txt")
  ff.write(titles[7])
  ff.close()
}
```

B

```
fun func() {
  FinishFile("out.txt").use {
    it.write(titles[5])
  }
}
```

C

```
fun func() {
  FinishFile("out.txt")
    .write(titles[2])
}
```

Duração: 1h30

ISEL, 29 de junho de 2022