Examen 3

Pregunta 1:

- a)

  Voy a seleccionar el lenguaje Kotlin empieza con K.

  - 1a) Kotlin tiene diferentes maneras de manipular objetos entre ellos tenemos :

    - Clases e interfaces : Kotlin posee clases e interfaces. Para declarar una clase basta con utilizar la palabra “class” seguido del nombre de la misma, y por default las clases en kotlin son publicas, pero soporta la creación de clases privadas , abstractas, open, inner y outer (estas dos ultimas son de clases anidadas). Adicionalmente hay clases con palabras claves data y enum con funcionales especificas. Las clases permiten crear métodos sobre ellas mismas y sus subtipos/subclases.

      Para declarar una interfaz basta la palabra “interface” seguido del nombre de la interfaz.

      una interfaz contiene métodos abstractos y no abstractos. Las interfaces permiten crean métodos sobre objetos que hereden de ellas.

    - Constructores: Las clases o interfaces de kotlin tienen distintos tipos de constructores. Las instancias de un objeto pueden ser creadas a través de un constructor primario (sin necesidad de declararlo) o secundarios (declarando un constructor interno para la clase) incluso se pueden tener varios constructores para un objeto.

```
// clase abstracta con constructor implicito

abstract class Lado(val a: Int, val b: Int) {
  fun cualquieraDeLosVertices() : Int {
    return a
  }
}

// interfaz que representa a grafos.

interface Grafo : Iterable<Lado> {

    // Retorna el número de lados del grafo
    fun obtenerNumeroDeLados() : Int

    // Retorna el número de vértices del grafo
```

```
    fun obtenerNumeroDeVertices() : Int

    /*
     Retorna los adyacentes de v, en este caso los lados que tienen
      como vértice inicial a v.
     Si el vértice no pertenece al grafo se lanza una RuntimeException
     */
    fun adyacentes(v: Int) : Iterable<Lado>

    // Retorna el grado del grafo
    fun grado(v: Int) : Int

    // Retorna un iterador de los lados del grafo
    override operator fun iterator() : Iterator<Lado>
}

/*
  Notese que utilizamos la palabra override para poder definir una definicion
  propia de la funcion iterator , esto es necesario, porque existe una definicion
  de iterator que es heredada de otra clase o interfaz.
*/

-----------------------------------------------------------------

// clase publica con un constructor explicito y que hereda de una interface
// llamada grafo.
public class GrafoDirigido : Grafo {

    var listaVertices = mutableListOf<Int>()
    var numVertices =  0
    var listaAdyacencias = Array(numVertices) {mutableSetOf<Int>()}
    var grafo = Array(numVertices) {mutableListOf<Arco>()}

  constructor(numDeVertices: Int) {
        this.numVertices = numDeVertices
        this.listaAdyacencias = Array(this.numVertices){mutableSetOf<Int>()}
        this.grafo = Array(this.numVertices) {mutableListOf<Arco>()}
        for (i in 0..this.numVertices-1) {
            this.listaVertices.add(i)
        }
   }
}

// Para instanciar esta clase GrafoDirigio basta con :
let instanceOfGraph = GrafoDirigido(20)
```

- Kotlin utiliza referencia predeterminada para las instancias de objetos a través de la palabra “this”. Es posible que al utilizar funciones miembros de una clase no es necesario declarar la palabra “this” pero esto es azúcar sintáctico.

```
// Siguiendo el ejemplo de la clase anteriormente definida.


// clase publica con un constructor explicito y que hereda de una interface
// llamada grafo.
public class GrafoDirigido : Grafo {
```

```
    var listaVertices = mutableListOf<Int>()
    var numVertices =  0
    var listaAdyacencias = Array(numVertices) {mutableSetOf<Int>()}
    var grafo = Array(numVertices) {mutableListOf<Arco>()}

  constructor(numDeVertices: Int) {
        this.numVertices = numDeVertices
        this.listaAdyacencias = Array(this.numVertices){mutableSetOf<Int>()}
        this.grafo = Array(this.numVertices) {mutableListOf<Arco>()}
        for (i in 0..this.numVertices-1) {
            this.listaVertices.add(i)
        }
   }
}

// Para instanciar esta clase GrafoDirigio basta con :
let instanceOfGraph = GrafoDirigido(20)
```

2a) En kotlin como mencionamos anteriormente para crear instancias de objetos bien sean clases, interfaces u objetos no definidos como clases, no se utiliza la palabra new, de hecho kotlin no contiene la palabra new, para crear instancias simplemente se llama a la clase o interfaz, por ejemplo, miClase( ) . Adicionalmente kotlin posee un recolector de basura el cual originalmente fue pensado e implementado mediante conteo de referencias debido a que en palabra de JetBrains es una implementación fácil de realizar, sin embargo, esta implementación ha generado un bloqueo en el desarrollo avanzado del lenguaje, por lo que JetBrains ha comenzado a trabajar en cambiar este modelo por uno de tracing y que acepte concurrencia.

3a) Kotlin tiene la capacidad de tener asociación estática y dinámica, sin embargo, tiene por default la asociación estática (porque las clases son definidas por defecto como finales) . Para cambiar el comportamiento de la asociación kotlin tiene palabras claves y construcciones especificas, como por ejemplos las clases privadas, open (con los override se puede realizar asociación dinámica) , final(aunque esto es default) y los companion objects.

4a) Kotlin posee polimorfismo paramétrico con una sintaxis muy similar al de Java. se representa al tipo general con “T”.

```
class city <T> (val cityOf : T)
```

Adicionalmente kotlin no acepta herencia multiple para clases, debido a que las clases tienen estados y lógica de inicialización , lo cual incluye efecto bordes. Sin

embargo si acepta la herencia multiple de interfaces, es decir, una clase puede heredar de varias interfaces simultáneamente.

```
interface CanSing {
  fun sing()
}

interface CanDance {
  fun dance()
}

class Artist : CanSing, CanDance {
  override fun sing() = "Can SING"
  override fun dance() = "Can Dance like a russian"
}
```

Para el manejo de varianzas, estas se resuelven manualmente y especificando cual debe usarse.

```
interface CanSing {
  fun sing()
  fun c()
}

interface CanDance {
  fun dance()
  fun c()
}

class Artist : CanSing, CanDance {
  override fun sing() { println("Can SING") }
  override fun dance() {println("Can Dance like a russian")}
  override fun c() {
    super<CanDance>.c()
  }
}
```