CI3641 – Lenguajes de Programación
Ana Santos.
17-10602.

EXAMEN #3

Pregunta 1: Lenguaje escogido en Kotlin

Respuesta (a.i):

Kotlin es un lenguaje orientado a objetos, por lo que cuenta con constructores, métodos, campos, visibilidad y tipos genéricos:

Los constructores dentro de las clases pueden ser primarios o secundarios, en el caso de ser primario solo tendrán uno, pero en el caso contrario pueden tener varios constructores. El constructor primario se considera el principal, ya que se encuentra en la parte cabecera de la clase y recibe los datos que explícitamente se necesitan par inicializar las propiedades del objetos; estos datos pueden ser declarados con val o var.

También, se pueden encontrar con los Bloques de inicialización dentro de las clases de Kotlin. Este bloque sirve para expandir la inicialización de las propiedades en el caso de que la clase tenga un constructor primario, por lo que es necesario hacer la aclaratoria de que si los parámetros de constructor se llaman igual a las propiedades estas serán diferenciadas al colocar la expresión *this* cuando se quiere acceder la propiedad de la instancia generada con el constructor.

Ahora, si se menciona la visibilidad del constructor principal es clave hablar de los modificadores que se le pueden añadir al mismo, *public, internal, protected* y *private,* para adecuarlo a las necesidades del programador y sirven también para variables, funciones, entre otros. Con estos modificadores se puede hacer accesible a cualquier parte, accesible sólo en el alcance actual, accesible en la clase actual y clases “hijas”, y accesible sólo en el módulo actual, respectivamente.

Los constructores secundarios se usan cuando la lista de argumentos del primario no satisface la creación del objeto en alguna situación. Este se declara dentro de la clase y es obligatorio usar la expresión *this* para delegarle los parámetros necesarios. Lo más interesante de estos constructores es que se les puede añadir un cuerpo que actúa “casi” como *init* condicionado por los argumentos del constructor.

Los métodos son un conjunto de instrucciones definidas dentro de una clase, que realiza una tarea y se puede invocar esta mediante un nombre. En Kotlin, los métodos son las funciones dentro de las clases y pueden contar con parámetros de entrada y/o de salida; además, cuentan con algo favorecedor a nivel de complejidad, la posibilidad de declarar una función en una sola línea.

En cuanto a la declaración de propiedades en las clases, tenemos que se pueden usar var si esta tendrá la oportunidad de ser modificada y val para solo la lectura del valor. Por otra parte, para los *getters y setters* son opcionales el inicializador, el obturador y el fijador, ya que el tipo de propiedad se puede deducir del inicializador.

Finalmente, los campos de apoyo no se pueden declarar dentro de las clases, sin embargo, en el caso de necesitarlo, Kotlin se encarga de proporcionarlo de manera automática. Si se quiere hacer referencia a este campo de referencia basta con usar el identificador de *field*. Esto sólo esta permitido usarlo en los descriptores de acceso a una propiedad, y en el caso de que no encaje en la implementación un campo de respaldo implícitamente, se puede crear una propiedad de respaldo.

Respuesta (a.ii):

El recolector de basura en Kotlinc depende de la configuración de la configuración del tiempo de ejecución de JVM y el mismo no tiene el mismo lenguaje que progtwig. Por eso, se puede decir que la gestión de memoria no necesita la intervención explicita del programador. Además, el recolector de basura asegura que el espacio no utilizado se limpie u la memoria se pueda liberar cuando no se necesite. La recolección reduce la carga del programador al realizar la asignación o la desasignación de memoria ydetiene la implementación de otros procesos, por lo tanto es costosa.

Respuesta (a.iii):

Kotlin usa la asociación dinámica de métodos, aunque simplifica la manera de definir métodos estáticos, usando las palabra *object* y funciones de alto nivel del paquete. Por otra parte, permite definir clases anidades forma estática por defecto y estas clases anidadas no contienen una referencia a la clase adjunta. En el caso de querer una clase anidada no estática, se debe usar la expresión *inner.* También, se puede declarar una función como estática usando la palabra *static*.

Hay que precisar que Kotlin limita los riesgos de fuga de memoria de manera predeterminada, al no existir la referencia que encierra a la clase por defecto.

Respuesta (a.iv):

El tipo más alto en la jerarquía de tipos se llama *Any*, es decir, todas las clases en Kotlin heredan de este tipo, incluyendo a *String, Int, Double* y así sucesivamente. Esta raíz contiene tres métodos que comparten su posición jerárquica *equals, toString* y *hashcode.* Otro tipo de jerarquía similar es *Nothing,* usado para funciones que retornan una excepción.

La herencia múltiple no es posible en Kotlin, ya que actúa como Java en este sentido, pero aunque no permiten heredar de varias clases, permite implementar vario "contratos", lo que hace que los métodos no puedan tener cuerpo sino que cuentan simplemente con la firma del método. Para lograr esto, Kotlin hace uso del *Protocol Oriented Programming (POP)*, el cual se encarga de simular la herencia múltiple usando contratos, preservando el estado de las instancias de las clases y dándole una extensión a los métodos definidos en la interfaz. Finalmente, un plus que tiene Kotlin es definir un cuerpo para cada método a comparación con otros lenguajes que usan POP.

En cuanto al polimorfismo general, Kotlin cuenta con él debido a que es similar al del Java, entendiendo que estos desaparecen a tiempo de ejecución. Además, tiene

métodos *inline* prometen al compilador que esta información está disponible usando *refied.*

Finalmente, el manejo de varianzas en Kotlin se usa varianza del sitio de statement, en el cual el parámetro de tipo T de la fuente asegura que solo se devuelve de los miembros fuentes, y nunca se consume. La varianza describe los valores que pueden pasar como argumentos y cuales esperar de los valores devueltos por los métodos cuanto cuando se trabaja con un tipo fuente/superclase. Por lo que, se puede tener de tipo de una clase T para devolver algo del mismo tipo, *out SomeType* para devolver un tipo o cualquiera de sus subtipo y si espera un *in SomeType* rspera alguna de los supertipos.

Pregunta 2: Lenguaje escogido PHP

Respuesta (a.i):

PHP posee una capacidad nativa mediante el uso de hilos, ya que por cada solicitud se invoca y se sirve un nuevo proceso de este lenguajes. Veamos que, PHP no brinda una funcionalidad de subprocesos múltiples propia, es necesario agregar los subprocesos como un paquete a nuestro proceso. Esto se puede realizar a través de 3 tipos de clases básicas con las que cuenta este lenguajes: *thread, worked* y *pool*. La primera clase trabaja como una unidad de instrucciones ejecutables de forma asíncrona, ejecutándose con la capacidad de un hilo. La segunda clase trabaja como un hilo y sincroniza las respuestas del proceso y la tercera maneja las múltiples instancias de la clase anteriormente mencionada y hacer su respectiva administración.

Respuesta (a.ii):

Como vimos en el inciso anterior, la clase *pool* es la encarga de manejo de tareas concurrentes, ya que es un contenedor y un controlador de *workers*. Su uso proporciona una abstracción en un nivel más alto que la funcionalidad de sus tareas, por lo que puede administrar la referencias en el orden necesario para *pthreads.* Esta clase se encarga de

usar un grupo de objetos inicializados para ser usados, es decir el cliente solicitará un objeto tipo pool y realizará operaciones en el objeto devuelto, al finalizar se obtiene de un objeto de tipo específico para el grupo del pool, en vez de destruirlo.

Los objetos del pool aumentan en gran nivel el rendimiento cuando el costo de inicializar una instancia de una clase y la creación de instancias de esta es alto, y cuando la cantidad de instancias en uso en ese momento es baja. Es importante, destacar que el *pooling* devuelve un objeto en un tiempo predecible.

PHP cuenta con dos interfaces para compartir memoria, *shm* y *shmop*. La primera intercambia variables, serializando y almacenando variables de objetos, aunque esto no funciona para funciones de acceso a operaciones. El segundo tiene funciones más débiles y se implementan haciendo un llamado a las funciones de la serie shm* para Linux y se hace la misma llamada encapsulando funciones del sistema.

Respuesta (a.iii):

Los scripts de PHP son inherentes e inmunes a errores provocados por sincronización interna, ya que no hay un soporte nativo para crear subprocesos que compartan las mismas variables, ejecutándose simultáneamente. Por otro lado, todo tipo de recurso compartido es candidato para problemas de sincronización; entendiendo que los scripts son de corta duración y no tienen estado, los recursos al mantener su estado se comportan como scripts diferentes o incluso llegan a ejecutar el mismo script más de una vez es un mismo momento. Finalmente, podemos decir que PHP tiene un comportamiento similar a Java, en cuanto a la sincronización e interferencia de sus subprocesos.