Estudiantes
10-10969 - Samuel Arleo R
11-10396 - Isaac Gonzalez

En esta primera sección se expondrán 9 aseveraciones que pueden ser ciertas, falsas o como han visto a lo largo del curso teórico, contraparte de este, pueden depender. Los paradigmas de programación, sistemas de tipos, alcances, asociaciones, así como otros aspectos, califican a un lenguaje pero esto podría, o no, limitar las capacidades del mismo. En ciertas ocasiones puede llegar a ser difuso pero la idea de esta primera parte de la asignación es que explore los posibles casos de la aseveraciones, donde podrían estar en lo correcto, dónde podrían no estarlo y hacia dónde se inclinaría la balanza. La idea es que el estudiante exponga su punto de vista como programador que ha trabajado con varios lenguajes, ha explorado distintos paradigmas y concluya con un sí, un no o depende.

Respuestas

> Los lenguajes de programación orientados a objetos que poseen herencia simple están limitados a incorporar comportamientos de un solo ancestro al momento de definir una clase.

http://stackoverflow.com/questions/3556652/how-do-java-interfaces-simulate-multiple-inheritance

Si, ya que para incorporar comportamientos de otras clases sería necesaria la herencia múltiple. Sin embargo es posible simular la herencia múltiple (sólo de métodos) por medio de interfaces si el lenguaje las provee. Esta técnica es en realidad un patrón de diseño llamado Delegación.

Por ejemplo, en Java podríamos definir una clase A que herede los comportamientos de otra clase B y que a su vez implemente una interfaz C. Esto permite simular que A hereda de una clase B y de otra C abstracta con los mismos métodos de la interfaz.

> Lenguajes de POO con un sistemas de tipos estático (C++, Java, C\#) no tienen la posibilidades de elegir la implementación de un método a tiempo de ejecución (despacho dinámico).

Depende. C++ posee un sistema de tipos estático pero ofrece tanto despacho estático como dinámico (declarando al método como virtual). Para esto utiliza una estructura de datos llamada tabla virtual que dada una clase mapea mensajes con métodos. Por lo tanto, al llamar un método declarado como virtual se escogerá el método según lo que almacene la tabla.

https://en.wikipedia.org/wiki/Dynamic_dispatch#Single_and_multiple_dispatch

> La introspección y reflexibidad son conceptos que se manejan en la POO pero no guardar

ninguna relación entre sí.

Es falso. La introspección es la capacidad de un programa de examinar el tipo o las propiedades de un objeto a tiempo de ejecución. La reflexión es la capacidad de un programa de examinar, hacer introspección y manipular los valores, la metadata y propiedades o funciones de un objeto a tiempo de ejecución. Por lo tanto, la introspección es una de las capacidades que ofrece la reflexión.

http://stackoverflow.com/questions/351577/why-is-reflection-called-reflection-instead-of-introspection?lq=1

http://stackoverflow.com/questions/351577/why-is-reflection-called-reflection-instead-of-introspection?lq=1

https://en.wikipedia.org/wiki/Reflection_(computer_programming)

https://en.wikipedia.org/wiki/Type_introspection

> En un lenguaje con un sistema de tipos dinámico la sobrecarga de métodos es innata y representa una comodidad dado que permite implementar un mismo método para distintos tipos.

Es verdadero. La sobrecarga normalmente se traduce en llevar a cabo acciones dependiendo del tipo del objeto, por lo que es necesario programar los distintos comportamientos del método según el tipo del objeto que llama al método o los parámetros. Python, por tener un sistema de tipos dinámico, permite utilizar el realmente el mismo método y definir distintos comportamientos dentro. Los lenguajes de tipado estático como Java realizan sobrecarga creando varios métodos con el mismo nombre y distintos argumentos, donde el lenguaje es capaz de diferenciarlos utilizando el número de argumentos del método o el tipo de los mismos (utilizando el contexto).

> En los lenguajes POO existen los términos interfaz, módulo, clase abstracta, rol, etc; definidos como objetos que pueden encapsular definiciones de clases o implementaciones concretas de métodos.

Depende de cómo esto se interprete. En las interfaces no es posible dar implementaciones de métodos. En cuanto a la encapsulación de definiciones de clases, los elementos de la POO mencionados permiten en realidad encapsular el funcionamiento y conformación de un objeto.

> Los métodos virtuales permiten asociar, al momento de compilar, una implementación de un método sobrecargado con una llamada del mismo; eliminando el **overhead** del despacho dinámico.

Los métodos virtuales son una forma de soportar el despacho dinámico. El objetivo de los métodos virtuales está relacionado con la elección entre métodos con el mismo

nombre definidos tanto en una subclase como en sus ancestros, no en la escogencia de un método de entre los que poseen el mismo nombre en la misma clase (sobrecarga de métodos). Sin embargo, es cierto que los métodos virtuales reducen el overhead que generaría calcular el método correcto a ejecutar, ya que lo hacen a tiempo de compilación.

La sobrecarga de tipos no produce despacho dinámico en C++, ya que el lenguaje considera el tipo de los parámetros como parte del mensaje.

https://en.wikipedia.org/wiki/Dynamic_dispatch (Ver parte de C++)

https://msdn.microsoft.com/en-us/library/aa645767(v=vs.71).aspx

https://en.wikipedia.org/wiki/Virtual_method_table

> Cuando un lenguaje de POO tiene herencia simple no ocurre el problema del diamante pero de igual forma pueden existir llamadas ambiguas de métodos, dado que incorporar interfaces, módulos, protocolos, etc, no evita colisión de nombres.

Las colisiones de nombres en interfaces no poseen importancia, ya que si un objeto A implementa dos interfaces IA e IB que definen métodos con el mismo nombre, el unico requisito es que A implemente el método, cumpliendo el contrato con ambas interfaces al hacerlo.

http://stackoverflow.com/questions/5164582/how-does-using-interfaces-overcome-the-problem-of-multiple-inheritance-in-c

> El paso de mensaje es un término que se maneja en modelos concurrentes, también de POO y es equivalente a la llamada de una función.

El término se emplea en ambos ámbitos. Sin embargo, no es lo mismo el paso de mensaje a una llamada de una función, ya que el primero genera un overhead adicional. Además, los argumentos son pasados de maneras distintas. En la llamada a función son pasados por medio de registros o en una lista de parámetros, mientras que en el paso de mensaje es necesario copiar cada argumento en una porción del mensaje.

https://en.wikipedia.org/wiki/Message_passing#Message_passing_versus_calling

http://programmers.stackexchange.com/questions/140602/what-is-message-passing-in-oo

> Sin importar la herencia del lenguaje de POO, una clase podría tener más de un ancestro.

Si, ya que la superclase en el nivel superior inmediato de la jerarquía podría a su vez ser una subclase.