**TP Integrador Final Paradigmas 2018 – Ruby**

* Disciplina de Tipado: cómo Ruby tipa las variables.

Ruby tiene un tipado dinámico y soporta polimorfismo de tipos, es decir, permite tratar a subclases utilizando la interfaz de la clase padre. Ruby no requiere de polimorfismo de funciones (sobrecarga de funciones) al ser dinámicamente tipado (los parámetros de una función pueden ser de distinto tipo en cada llamada, encargándose dicha función de determinar el tipo y actuar en consecuencia).

* Interpretación: ¿Es ruby un lenguaje compilado o interpretado?

Ruby es un lenguaje interpretado debido a que el código fuente es convertido a lenguaje de máquina a medida que es ejecutado.

Al ser Ruby un lenguaje interpretado, permite el tipado dinámico de datos, por lo tanto no es necesario inicializar una variable con determinado tipo de dato.

* Metaprogramación: ¿Qué significa metaprogramar? ¿Qué ofrece Ruby en este sentido?

Metaprogramar consiste en escribir programas que escriben o manipulan otros programas (o a sí mismos) como datos, o que hacen en tiempo de compilación parte del trabajo que, de otra forma, se haría en tiempo de ejecución. Esto permite al programador ahorrar tiempo en la producción de código.

Ruby brinda métodos útiles para utilizar en la metaprogramación:

* Introspección o reflexión: en ruby es posible leer información sobre una clase u objeto en tiempo de ejecución. Para esto se pueden usar los siguientes métodos: class(),instance\_methods(), instance\_variables().
* Send: send() es un método de instancia de la clase Object.
* define\_method() permite realizar la definición de un método dentro de la clase de manera dinámica y en tiempo de ejecución.
* method\_missing() es un método llamado por Ruby cuando algún método no es encontrado durante el method lookup.
* remove\_method() y undef\_method(), el primer método elimina un método de una clase en particular, mientras que el segundo evita que la clase especificada responda a las llamadas al método nombrado.
* Clases abiertas: ¿Qué son las open clases o clases abiertas? ¿Qué ventajas y desventajas tienen? ¿Qué es el monkey patching?

En Ruby las clases están abiertas, por lo tanto se pueden abrir, agregarles funcionalidad y modificarlas en cualquier momento. Esto se aplica tanto a las clases que uno escribe así como también a las estándar incluidas en Ruby.

La ventaja del uso de las clases abiertas es poder realizar modificaciones al escribir código.

La desventaja es que el poder cambiar el resultado esperado de un método puede causar comportamientos extraños y puede ser difícil encontrar los errores.

Monkey patching es una herramienta que permite realizar modificaciones en una clase o módulo en tiempo de ejecución para poder solucionar un error o característica que no actúa como se desea.

* Módulos: ¿Qué es y qué tres usos tiene un module? ¿Qué es un mixin? ¿Para qué sirve?

Un módulo es una colección de métodos, constantes y clases.

Actúan como librería, lo que permite definir métodos cuyos nombres no coincidirán con los definidos en otros sitios.

Además permiten aumentar la funcionalidad entre clases: si una clase usa un módulo, los métodos de ese módulo estarán disponibles como si se hubieran definido en la clase.

Un mixin brinda una manera controlada de agregar funcionalidad a las clases. Sin embargo, su verdadero poder está cuando el código en el mixin empieza a interactuar con el código de la clase que lo usa.

* Pureza: ¿se cumple el “todo es un objeto”? ¿Qué cosa no lo es? ¿Qué consecuencias tiene?

En Ruby “casi todo es un objeto”, es decir, “casi” todo tiene una serie de propiedades (variables de instancia) y sobre el objeto se pueden realizar acciones (o ejecutar métodos)

Los mensajes no son objetos, tampoco los condicionales, bucles y los bloques.

Esto tiene como consecuencia tener que realizar más acciones si se los quiere convertir en objetos. Por ejemplo, para convertir los bloques en objetos se puede usar Proc, lambda o el constructor literal ->.

* Objetos: ¿se pueden declarar objetos “al vuelo” y darles comportamiento dinámicamente?

No, en Ruby no se pueden declarar objetos “al vuelo”. Los objetos se crean a partir de clases; éstas sirven como moldes para crear objetos. Al crear un objeto de una clase, se crea una instancia de la clase.