¿Una comparación entre Java y .NET?

Luz Elena Jiménez
Universidad Icesi
ljimenez@icesi.edu.co

Fecha de recepción: 10-2-2003 Fecha de aceptación: 12-8-2003

RESUMEN

En este artículo se presentan una serie de reflexiones frente a las comparaciones que pueden hacerse entre dos plataformas de software: Java y .NET.

Para ello se trata de hacer un breve recuento histórico de ambos casos, y después se presentan algunas de las diferencias que la autora ha encontrado entre ellas, mirando aspectos que tienen relación directa con la programación orientada a objetos, o con otros aspectos del lenguaje. Por último se presenta una breve aclaración, desde el punto de vista de la autora, frente al tema de portabilidad que ambos reclaman como la diferencia más relevante entre ellos.

PALABRAS CLAVES

Programación orientada a objetos, Java, .NET, C#.NET.

SUMMARY

It in this article we find a series of reflections and comparisons that could occur between the two platforms of software: Java and .NET.

For that reasons I have tried to give a brief historic account of both cases, and later some of the differences that the author has found between them are introduce, looking at the aspects that have direct relationship with the programming oriented by objects, or with other areas of the language.

Finally a brief explanation with the point of view of the author is introduced, the topic referring to portability that both Java and .NET claim to have.

KEYWORDS
Programming Oriented by objects, Java, .NET, C#.NET

Clasificación: B

INTRODUCCIÓN

Inicialmente el título de este artículo fue "Una comparación entre Java y .NET"[1], sin embargo al empezar a desarrollarlo surgieron dudas frente a qué comparar. La verdad es que no es sencillo dada la complejidad de los elementos que quieren compararse; inicialmente puede pensarse que se trata de comparar dos lenguajes de programación pero muy rápido se da uno cuenta de que no es así de fácil pues en ambos casos hay mucho más.

Pero hay otro problema, y es que conocer realmente bien ambos "elementos" es muy complicado, así que se debe buscar bibliografía especializada y revisar cuidadosamente los puntos expuestos, pues la mayoría de las publicaciones alaban a Java y atacan a .NET o alaban a .NET y atacan a Java. Y a partir de esto es conveniente dejar en claro desde ahora que este artículo no buscará encontrar un vencedor: se reconoce que en ambos casos hay puntos a favor y puntos en contra, y que por tanto no puede hallarse un ganador absoluto en una comparación.

Así que lo que se tratará de hacer a lo largo de este artículo será:

- Una breve presentación de Java y su historia.
- Una breve presentación de .NET y su historia.
- Comparar lenguajes de programación: Java vs. C#.NET, C++.NET y VB.NET[2]
- Comparar la portabilidad en ambos casos.

Por último, antes de entrar en materia, se quiere dejar presente que este artículo se escribe partiendo de que quien lo lee tiene conocimientos de programación orientada a objetos, por ello no se entra a explicar ninguno de los términos relacionados, y también que existe un conocimiento previo de Java, dado que en el artículo se presentará código elaborado en .NET y las diferencias no serán claras para quien desconozca Java.

Java: Su historia

Java nace como un lenguaje de programación fácil de utilizar: algunos de los programadores de Sun Microsystems, cansados de "pelear" con C++, obtienen la autorización de sus jefes para desarrollar un lenguaje de programación que sea muy sencillo de utilizar, y que pueda ser ejecutado sobre dispositivos pequeños: televisores, electrodomésticos, o cualquier implemento eléctrico instalado en casa. Este proyecto empieza a demandar tiempo y dinero, y se obtiene un primer producto, OAK,[3] con el cual no se logra la aceptación esperada, y el proyecto queda descartado. Un tiempo después, con la aparición de

1. A lo largo del artículo se menciona primero Java, simplemente porque existe hace más tiempo.
2. Por facilidad, en este artículo, se referenciarán como .NET
3. En inglés, roble.

WWW, se piensa que el proyecto debe ser revaluado, y se le da un nuevo impulso, apareciendo Java.[4]

Pero, ¿qué es lo que se presenta como Java? Lo que Sun presenta al mundo como su nuevo lenguaje de programación es algo más: es un lenguaje de programación orientado a objetos, que además incluye una máquina virtual, y una serie de desarrollos básicos que pueden ser empleados por los programadores para simplificar sus nuevos desarrollos; además permite la inclusión de porciones de código ejecutables en las páginas que se publican en internet, a través de WWW.

Sun permite también que otros fabricantes de software tomen a Java como el centro de sus nuevas herramientas de software, es decir, desarrollen entornos de programación considerando a Java como su corazón; de manera que en un tiempo relativamente corto IBM, Borland, Oracle y muchos otros (incluyendo a Microsoft) están desarrollando herramientas que les permitan ofrecer productos portables entre diferentes máquinas y sistemas operativos, pues todos compilan para la misma máquina: la máquina virtual de Java.

Hoy en día, Java cuenta con un API amplio, en el cual pueden encontrarse puntos de partida para crear los más diversos tipos de programas, tiene una serie de estándares para programación reconocidos y aceptados por la mayoría de sus programadores, existen las versiones de su máquina virtual para casi todas las plataformas comerciales vigentes en este momento y está en capacidad de hacer interfaz con casi todas las bases de datos existentes en el mercado. Además, Java tiene otra gran ventaja: Microsoft le declaró la guerra hace algún tiempo y muchas de las personas involucradas con el negocio del software que no aceptan a esta empresa ven a Java como su aliado.

A partir de lo presentado hasta el momento se puede creer que Java es un lenguaje de programación bastante popular, y en lo que a internet se refiere esto es cierto, pues tiene buena parte de los servidores conectados a esta red. Sin embargo no todo es dicha, pues en la parte de aplicaciones comerciales para las empresas no se ha logrado tener el mercado ni la aceptación que se quisiera, y las razones son varias: Java no ofrece un entorno de programación completo, además el producto final no es muy veloz en tiempo de ejecución, entre otras cosas.

.NET: Su historia

Prácticamente desde la aparición de Java, Microsoft ha querido ser su competencia, para ello inicialmente firmó algunos convenios con Sun, para trabajar con Java igual que lo estaban haciendo IBM y Oracle, pero en realidad trató de crear su versión propia de Java, denominada J++, la cual no era completamente compatible con la versión estándar. Además introdujo algunas modificaciones en sus sistemas operativos y navegadores que hacían que el desempeño de

4. En inglés, forma coloquial de llamar al café, y quienes crearon el lenguaje quisieron rendirle un homenaje a la que fuera su bebida favorita. Esto también explica la presencia de la humeante taza en casi todas las publicaciones referentes al lenguaje.

aplicaciones desarrolladas en Java fuese más lento de lo que debía.

Pero en el momento en que Java empieza a consolidarse como el lenguaje de internet, la decisión es más fuerte, hay que lanzar "algo" que realmente le haga contrapeso a Java, y que pueda ser considerado como la competencia de Java en internet, y a raíz de ello aparece .NET.

Y aquí hay que tratar de revisar qué es .NET, inicialmente se podría pensar que es un lenguaje de programación orientado a objetos, o un conjunto de lenguajes de programación todos ellos orientados a objetos, y se pueden citar C#.NET, C++.NET y VisualBasic.NET; pero a esta última definición le hace falta incluir lo referente al acceso a las bases de datos, es decir ADO.NET, las herramientas para desarrollo en internet, ASP.NET y el conjunto de facilidades para construir Servicios Web, además del hecho de que todo viene integrado dentro de un entorno completo denominado VisualStudio.NET.

Para lograr elaborar estas herramientas y tener la certeza de que se integrarán bien, se definieron los siguientes elementos:

- Un lenguaje común de ejecución: CLR (Common Language Runtime).
- Un conjunto de tipos de datos básicos: CTS (Common Type System), el cual incluye, además de todos los tipos de datos básicos, las clases Object y String.
- Un CLS (Common Language Specification), que es el conjunto de reglas que especifica lo referente a la implementación de las características de la POO y a otras estructuras sintácticas.
- Un MSIL (MicroSoft Intermediate Language), o lenguaje intermedio común. Este lenguaje es el equivalente al bytecode de Java.
- Un compilador capaz de traducir del MSIL a lenguaje binario, comúnmente denominado JIT (Just In Time).

Así pues, se puede decir que .NET es más que un conjunto de herramientas: con .NET se rompe la filosofía tradicional de Microsoft, pues se ha hecho un esfuerzo para que .NET sea abierto y estándar; para lo cual sometieron sus especificaciones del lenguaje y su tipo común de datos a la revisión de organismos internacionales dedicados a la regulación y estandarización de las plataformas de programación;[5] y una vez aprobados los hicieron públicos. De hecho Microsoft garantiza que si otro productor de software construye un lenguaje que al compilar lleve al tipo común de datos y respete las especificaciones que ellos han dado, ese lenguaje podrá ser incorporado a ambientes .NET, y las clases creadas a través de él podrán interactuar con los elementos de .NET sin ningún problema. Y como muestra de esto puede hacerse referencia a "Mono", de Ximian, que se esfuerza en ofrecer una implementación para cualquier tipo

5. Estas especificaciones están avaladas por ECMA : http://www.ecma-international.org/

de ambiente Unix, de todo el Framework de .NET, y ya ofrece el compilador para C#, el runtime (compilador, intérprete, recolector de basura, manejador de multi-hilo, etc.), versiones del API, además de versiones de ADO.NET y de ASP.NET.[6]

Java vs. los lenguajes de .NET

Siguiendo con lo propuesto se van a presentar algunas diferencias entre los lenguajes de .NET y Java, vistos como lenguajes de programación orientados a objetos, para lo cual se partirá de ver la forma en la cual cada uno de ellos implementa las características de la POO, cómo es el manejo de los objetos como tales, el encapsulamiento, la herencia y el polimorfismo. Y también se verán algunas diferencias de sintaxis.

Tipos de estructuras que el lenguaje le permite crear a un programador:

Java	.NET
Clases e interfaces.	Clases, interfaces, struct y enum.

Elementos que pueden definirse dentro de una clase:

Java	.NET
Atributos, métodos y clases internas.	Atributos, métodos, clases internas, propiedades, eventos y delegates.

Niveles de encapsulamiento:

Java	.NET
public, private, protected y visibilidad de paquete, este último se asume cuando se omite.	public, private, internal, protected y la combinación de estos dos últimos. En caso de omisión se asume private.

Herencia:

Java	.NET
No se permite la herencia múltiple, se puede simular a través del uso de interfaces. Por omisión se hereda de Object.	No se permite la herencia múltiple, se puede simular a través del uso de interfaces. Por omisión se hereda de Object.

6. http://www.go-mono.com/

Polimorfismo:

Java	.NET
Se permite que una clase sobrecargue o sobreescriba métodos definidos por su clase padre, a menos que la clase padre lo impida mediante la palabra reservada *final*, en el encabezado del método. Si un objeto de una clase hija es referenciado a través de una referencia a su clase padre, su comportamiento, al invocar un método sobreescrito, será el que definió la clase a la cual él pertenece.	Se permite que una clase sobrecargue o sobreescriba métodos definidos por su clase padre, a menos que la clase padre lo impida empleando la palabra reservada *sealed*, en el encabezado del método. Si un objeto de una clase hija es referenciado a través de una referencia a su clase padre, su comportamiento, al invocar un método sobreescrito, dependerá de los permisos establecidos por la clase padre, y de la decisión tomada por quien definió la clase. Por omisión se comportará como lo definió la clase padre.

Sobrecarga de operadores para una clase:

Java	.NET
No permite la sobrecarga de ninguno de los operadores básicos.	Se permite la sobrecarga de algunos de los operadores básicos: Unitarios: **+**, **-**, **!**, **~**, **++**, **--**, **true**, **false** Binarios: **+**, **-**, ***, /, %, &**, **|**, ^, <<, >>, ==, **!=** , >, <, >=, <=

Ahora se entrarán a discutir más a fondo, algunos de los tópicos presentados en las tablas anteriores.

El que .NET permita la creación de varios tipos de estructuras de datos, y no sólo clases e interfaces como Java, da la posibilidad de que en este ambiente se puedan crear tipos de valores. Y al respecto hay algunas particularidades que es interesante revisar, pues cada nuevo tipo, siempre que se trate de un *struct*, se define como descendiente de una clase determinada, es decir hereda de Object, y por tanto puede definir atributos, propiedades, métodos, eventos o delegates, incluso implementar interfaces y definir clases internas, pero el comportamiento de sus instancias será el de valores, no el de objetos. Estas estructuras tienen dos restricciones, no pueden tener clases hijas y no pueden sobreescribir el constructor sin parámetros.

También hay diferencias grandes en lo referente a aquellos elementos que

pueden definirse dentro de una clase[7] en .NET, respecto a los que pueden definirse dentro de una clase en Java. .NET permite los mismos elementos que Java, pero adiciona las propiedades, que tienen un comportamiento muy similar a las propiedades que se emplean en los beans en Java, pero que en este ambiente son fundamentales, dado que aquí se acostumbra que los atributos sean definidos como privados, es decir el máximo nivel de encapsulamiento, y que el programador defina la forma de hacer una asignación sobre ellos o conocer el valor que contienen, mediante estas propiedades. Aquí es importante resaltar el hecho de que las propiedades no necesariamente deben estar asociadas a un atributo, pueden estar asociadas a varios o a ninguno, además tienen la posibilidad de contener un método get y otro set, pero pueden omitir cualquiera de los dos. Por último estos métodos no pueden ser invocados directamente, sino que se trabaja como si se tratara de asignaciones corrientes, es decir, si en una clase se tienen estas definiciones:[8]

```
. . .
int valor; // se define un atributo privado
double porcentajeDescuento = 0.10; // se define un atributo privado
public int ValorBruto // se define una propiedad {
set {
        valor = value; }
get {
        return valor; }
}
public double ValorNeto // se define otra propiedad {
get {
        return valor * (1 - porcentajeDescuento); }
}
```

y en otra clase se crea un objeto perteneciente a esta clase, y se usan estas propiedades:

```
unObjeto.ValorBruto = 5; // se asigna 5 al atributo valor, a través
                // de la propiedad ValorBruto
Console.WriteLine(«El valor del artículo, sin descuento es «+unObjeto.ValorNeto);
Console.WriteLine(«Y el valor, después de aplicar el descuento es «+unObjeto.ValorBruto);
// en las dos líneas anteriores se muestra el valor que retornan las propiedades ValorNeto
// y valor Bruto.
```

7. En adelante se empleará el término «clase», para referirse tanto a ellas, como a los struct.

8. Todo el código que se incluye en este artículo se ha elaborado en C#.NET

También se presentan diferencias en lo referente al manejo de los eventos, pues mientras que en Java estos son objetos que pertenecen a la jerarquía de clases de EventObject; en .NET los eventos son un elemento más que define dentro de una clase. En Java, quien desea escuchar un evento debe implementar una interfaz que lo identifique como oyente del mismo, mientras que en .NET, en la definición del evento debe incluirse quién lo manejará, y a estos manejadores se les denomina delegates, y también debe especificarse el método o los parámetros que requiere para atender la situación cuando ocurra el evento. Es decir, que en la definición de una clase bien puede aparecer algo así:

```
public delegate void ManejadorDeDesastres(string alerta);
/** se define quién se va a encargar de atender la ocurrencia de los eventos, y qué recibe como parámetro */
public static event ManejadorDeDesastres Terremoto;
public static event ManejadorDeDesastres Incendio;
/** se definen dos eventos, y se asocian ambos al mismo manejador de eventos */
```

También se presentan diferencias en el manejo de los niveles de encapsulamiento, pues .NET asume que el nivel de encapsulamiento por defecto es privado, es decir, cualquier cosa que se defina, y cuyo modificador de acceso se omita se asume fue definida como privada. Además, maneja un nivel más de encapsulamiento, que permite a una clase definir que da permiso de acceso sobre un elemento determinado a todas las clases que hereden de ella, independiente del espacio de nombre —que es lo mismo que los paquetes en Java— en el cual se encuentren definidas. El modificador que se emplea para manejar este nivel es protected, o sea que aquí este modificador no obra como en Java; y si se quiere tener el mismo nivel de encapsulamiento que da este modificador en Java, se debe emplear la combinación protected internal, que es la única combinación de modificadores de acceso que .NET permite.

Pero probablemente una de las más grandes diferencias se refiere al comportamiento de los métodos sobreescritos cuando son solicitados a través de una referencia a la clase padre, pues dependiendo de la forma en la cual se haya hecho la definición del método, tanto en la clase padre como en la clase hija, el comportamiento variará. Esto se ilustra con las siguientes porciones de código:

- La clase padre niega el permiso de modificar el comportamiento al ser referenciado a través de ella:

```
public class Personas {
        public void MuestraMensaje() {
                Console.WriteLine("Estoy en la clase padre");
  }
}
public class Hijos:Personas { // Indica que hereda de Personas
        public void MuestraMensaje() {
                Console.WriteLine("Estoy en la clase hija");
  }
}
public class Gestión {
        public static void Main(String[] args) {
                Personas padre = new Personas();
                Personas hijo = new Hijos(); // un objeto de la clase hija asociado a una referencia
                                             // a la clase padre.
                padre.MuestraMensaje();
                hijo.MuestraMensaje();
  }
}
```

En este caso se mostrará en pantalla:

Estoy en la clase padre

Estoy en la clase padre

- La clase padre otorga el permiso de modificar el comportamiento al ser referenciado a través de ella, pero la clase hija no lo emplea:

```
public class Personas {
        public virtual void MuestraMensaje() {
                Console.WriteLine("Estoy en la clase padre");
  }
}
public class Hijos:Personas {
        public new void MuestraMensaje() {
                Console.WriteLine("Estoy en la clase hija");
  }
}
public class Gestión {
        public static void Main(String[] args) {
                Personas padre = new Personas();
                Personas hijo = new Hijos();
                padre.MuestraMensaje();
                hijo.MuestraMensaje();
 }
}
```

En este caso, también, se mostrará en pantalla:

```
Estoy en la clase padre
Estoy en la clase padre
```

Aquí es interesante agregar que si la clase Hijos tiene clases que hereden de ella, en ellas este método no podrá ser sobreescrito, a menos que se añada la palabra *virtual* antes del *new*.

- La clase padre otorga el permiso de modificar el comportamiento al ser referenciado a través de ella, y la clase hija lo emplea:

```
public class Personas {
        public virtual void MuestraMensaje() {
                Console.WriteLine("Estoy en la clase padre");
}
}
public class Hijos:Personas {
        public override void MuestraMensaje() {
                Console.WriteLine("Estoy en la clase hija");
}
}
public class Gestión {
        public static void Main(String[] args) {
                Personas padre = new Personas();
                Personas hijo = new Hijos();
                padre.MuestraMensaje();
                hijo.MuestraMensaje();
}
}
```

En este caso se mostrará en pantalla:

```
Estoy en la clase padre

Estoy en la clase hija
```

En este caso si la clase Hijos tiene clases que hereden de ella, este método podrá ser sobreescrito.

- La clase padre niega el permiso de modificar el comportamiento al ser referenciado a través de ella, y la clase hija intenta modificarlo:

```
public class Personas {
        public void MuestraMensaje() {
                Console.WriteLine("Estoy en la clase padre");
 }
}
public class Hijos:Personas {
        public override void MuestraMensaje() {
                Console.WriteLine("Estoy en la clase hija");
 }
}
public class Gestión {
        public static void Main(String[] args) {
                Personas padre = new Personas();
                Personas hijo = new Hijos();
                padre.MuestraMensaje();
                hijo.MuestraMensaje();
 }
}
```

En este caso se generará un error en tiempo de compilación.

En Java, los métodos siempre se comportan como si la clase padre hubiese empleado la palabra virtual en sus encabezados, y cada clase hija empleará el override al sobrecargarlo. Esto puede ser más sano, pues permite que un programador pueda crear clases que hereden de clases elaboradas por otros, y a cuyo código fuente no tenga acceso, sin preocuparse por este tipo de verificaciones, es decir empleando el polimorfismo como una característica natural de la programación orientada a objetos, mientras que en .NET si la clase de la cual se quiere heredar no otorga este permiso, o si el programador no conoce este tipo de limitantes, se pueden obtener resultados no deseados en un momento determinado.

Y la última diferencia que presentó en las tablas anteriores, es la sobrecarga de operadores, pues .NET permite que una clase defina el comportamiento que espera se dé cuando se empleen algunos de los operadores que ellos definen como "sobrecargables". Respecto a esta operación la única consideración que debe hacerse es que .NET puede exigir la sobrecarga del operador antagónico, y que todos los operadores asociados quedan sobrecargados. Por ejemplo, si se sobrecarga el operador de la suma (**+**), .NET exige la sobrecarga para el operador de la resta (**-**), y queda sobrecargado el operador que permite el incremento de un objeto (**+=**).

También hay diferencias en el paso de parámetros a un método, pues en Java todo el paso de parámetros se hace por valor, mientras que .NET permite varias formas de realizar este paso de parámetros siendo las más

comunes el paso por valor, el paso de parámetros por referencia, y el paso de parámetros cuya cantidad se desconoce, para lo cual se emplea el modificador *params*, el cual puede combinarse con un paso por valor o por referencia. Otra cosa que es importante tener en cuenta es que cada uno de los parámetros debe pasarse indicando la forma en que se pasa, es decir un método puede recibir unos por valor y otros por referencia. Además, no basta con que el encabezado del método indique cómo lo recibe, al invocar el método debe indicarse la forma en la cual se pasa. Por omisión se asumen pasos por valor.

Hay otras diferencias entre ambos lenguajes, dadas por la sintaxis, o por la aparición de algunas instrucciones que no existen en Java, o por algunos pequeños cambios en algunas otras. La instrucción *foreach* ilustra la aparición de nuevas instrucciones, pues es una instrucción repetitiva, que recorre cualquier colección de principio a fin, pasando por cada uno de los elementos que la conforman.

Para ilustrar la modificación de instrucciones, se puede mirar el caso de las excepciones, donde se encuentran dos diferencias, la primera es que el *catch* que no especifica qué tipo de excepción quiere atrapar, es equivalente al *catch* (*Throwable e*) en Java; y la segunda diferencia es que .NET no soporta el *throws* en el encabezado del método, es decir no se permite que las excepciones comprobadas se propaguen; lo cual puede causar inconvenientes, pues si se trabaja con un esquema en el cual las clases que dan estructura a la solución de un problema son independientes de las clases que se emplean para crear la interfaz de usuario, el poder propagar la excepción y permitir que el manejo del mensaje de error lo haga directamente la clase que brinda la interfaz no es fácil de hacer en .NET, y sería lo ideal.

Otro aspecto en el cual se presentan diferencias entre ambos lenguajes, es en lo referente al uso de hilos, pero no tanto en la forma de usarlos sino en lo que se usa. En Java, cuando se trabaja con varios hilos se pide que se creen procesos livianos con una serie de características, como las prioridades, pero dependiendo del sistema operativo con que se esté trabajando se pueden obtener resultados diversos, pues Windows no respeta las prioridades que se otorgan desde Java, pero Unix sí lo hace. Cuando se trabaja con cualquiera de los lenguajes de .NET y se hace trabajo multihilo, se está trabajando con los hilos del sistema operativo; pues por la forma en la cual fue construido .NET y dado el nivel de acceso que tienen sus creadores al núcleo del sistema operativo sobre el cual va a ejecutarse, ello permite que los programadores puedan emplear elementos contenidos en algunos dll de este sistema operativo, siendo el que maneja los hilos uno de ellos. Esto hace que el desempeño de los hilos en .NET sea más ágil que en Java, además permite que las características que el programador asigne a cada hilo sean respetadas por el sistema operativo; pero tiene el inconveniente de que un programador inexperto bien puede causar algunos daños al desempeño del equipo e incluso al sistema operativo.

Algunos autores presentan, como diferencia, el uso de apuntadores en

.NET, la verdad es que su utilización real es más bien restringida, pues sólo pueden emplearse dentro de la memoria administrada por .NET, requiere el uso de instrucciones especiales mediante las cuales el programador indica que está trabajando con código inseguro, y que quiere hacerlo; y no es equivalente al uso de estos en lenguajes como C o C++, pues también hay restricciones frente al tipo de operaciones que pueden ejecutarse. O sea que bien podría decirse que este uso de apuntadores en .NET tiene las suficientes restricciones como para que no se usen.

Otro aspecto que suele presentarse como diferencia entre ambas plataformas es el que se refiere al API de cada una. La verdad es que Java ofrece un API más grande y maduro que el que viene con .NET, sin embargo, y dada la aceptación que parece estar teniendo esta plataforma en el mercado es bastante probable que, al menos en lo que a cantidad se refiere, esta situación cambie en muy poco tiempo.

Portabilidad en ambos casos

La portabilidad es presentada, por los defensores de cada una de las plataformas, como una fortaleza suya y una debilidad del otro. La verdad es que ambas plataformas ofrecen portabilidad, pero en dos ámbitos completamente distintos.

Java ofrece portabilidad entre máquinas, es decir no depende ni del Hardware ni del sistema operativo. Y eso es una gran ventaja; sin embargo si un programador que trabaja en Java se encuentra con que quiere incluir en su código algunas funcionalidades elaboradas en C, por ejemplo, el proceso es complicado. Lograr que el código elaborado en Java pueda ser incluido en desarrollos elaborados en otros lenguajes tampoco es un trabajo sencillo.

.NET no tiene esta dificultad, la interacción entre código elaborado en cualquiera de los lenguajes que incluye esta herramienta es inmediata, como lo será con cualquier lenguaje que sea elaborado cumpliendo con las especificaciones del CLS y que lleve sus datos al CTS, al menos en la teoría. Pero esta plataforma, que ofrece una gran portabilidad en lo que a lenguajes de programación se refiere, está bien asociada a un tipo de sistema operativo específico: el que produzca Microsoft, al menos en el momento actual.

Cada una de ellas ofrece un tipo de portabilidad del cual la otra carece y es esa diferencia la que se intenta presentar como ventaja o desventaja en cada caso; cuando el análisis real para identificar la portabilidad de cada una depende de la situación específica y del alcance que se espera tener con el software a desarrollar.

CONCLUSIONES

Es indudable que Java cuenta con mayor madurez, con más programadores, con el respaldo de muchas casas de software, entre otras cosas que pueden considerarse ventajosas; pero no es menos cierto que .NET ha sido desarrollado por algunas de las personas que más conocen de Java, que cuenta con el respaldo de una inmensa casa de software, y con el apoyo de muchas personas en el mundo, además de tener una gran aceptación comercial.

Tal como se dijo al inicio de este artículo, la autora no cree que se pueda realizar una comparación generalizada, e imparcial, que permita encontrar un vencedor. Al compararlas, es fácil hallar ventajas y desventajas de una frente a la otra, así que quien esté tratando de tomar una decisión frente a cuál de las dos conviene más a su trabajo futuro, debe revisar estas características frente al que será su entorno de trabajo y con base en eso tomar una decisión.

BIBLIOGRAFÍA

http://www.ecma-international.org/

http://www.go-mono.com/

CURRÍCULO

Luz Elena Jiménez Collazos, Ingeniera de Sistemas de la Universidad Icesi. Trabajó en diversas empresas de la región en el área de Análisis y Desarrollo de Software, como Analista Junior, Senior y Jefe del Departamento de Sistemas. Y en el área de Redes como Ingeniera de Proyectos. Profesora de la Universidad Icesi en los cursos de Sistemas Operativos, Ingeniería de Procesos y Algoritmos; vinculada de tiempo completo.