C# vs. Java: Major Similarities

By Yoshitaka Shioutsu

The origins of both Java and C# are closely tied to the transition from lower-level programming languages like C++ to higher-level programming languages that compile into bytecode that can be run on a virtual machine. This comes with a number of benefits, most notably the ability to write human readable code once that can run on any hardware architecture that has the virtual machine installed on it. Syntactic quirks aside, it's not surprising how similar these two languages are from the top-level perspective of an application developer. Here are a few of the main similarities between C# and Java:

- **Type-Safe.** A type error occurs when the data type of one object is mistakenly assigned to another object, creating unintended side effects. Both C# and Java make a good effort to ensure that illegal casts will be caught at compile time and exceptions will be thrown out at runtime if a cast cannot be cast to a new type.
- **Garbage Collection.** In lower-level languages, memory management can be tedious because you have to remember to properly delete new objects to free up resources. That's not the case in C# and Java, where built-in garbage collection helps prevent memory leaks by removing objects that are no longer being used by the application. While memory leaks can still occur, the basics of memory management have already been taken care of for you.
- **Single Inheritance.** Both C# and Java support single inheritance—meaning only one path exists from any base class to any of its derived classes. This limits unintended side effects that can occur when multiple paths exist between multiple base classes and derived classes. The diamond pattern is a textbook example of this problem.
- **Interfaces.** An interface is an abstract class where all methods are abstract. An abstract method is one that is declared but does not contain the details of its implementation. The code governing any methods or properties defined by the interface must be supplied by the class that implements it. This helps avoid the ambiguity of the diamond pattern, because it's always clear which base class is implementing a given derived class during runtime. The result is the clean, linear class hierarchy of single inheritance combined with some of the versatility of multiple inheritance. In fact, using abstract classes is one way multiple inheritance languages can overcome the diamond pattern.

C# vs. Java: Major Differences

As similar as the two languages are in terms of purpose, it's important to remember that C# holds its origins in Microsoft's desire to have a proprietary "Java-like" language of their own for the .NET framework. Since C# wasn't created in a vacuum, new features were added and tweaked to solve issues Microsoft developers ran into when they initially tried to base their platform on Visual J++. At the same time, Java's open-source community continued to grow, and a technical arms race developed between the two languages. These are some of the major differences between C# and Java.

- **Windows vs. Open-Source.** While open-source implementations exist, C# is mostly used to develop for Microsoft platforms—the .NET Framework's CLR being the most widely used implementation of the CLI. On the other end of the spectrum, Java has a huge open-source ecosystem and gained a second wind in spite of its age, thanks in part to Google adopting the JVM for Android.
- **Support for Generics.** Generics improve compiler-assisted checking of types largely by removing casts from source code. In Java, generics are implemented using erasures. Generic type parameters are "erased" and casts are added upon compilation into bytecode. C# takes generics even further by integrating it into the CLI and allowing type information to be available at runtime, yielding a slight performance gain.
- **Support for Delegates (Pointers).** C# has delegates which essentially serve as methods that can be called without knowledge of the target object. To achieve the same functionality in Java, you need to use an interface with a single method or some other workaround that may require a nontrivial amount of additional code, depending on the application.

- **Checked Exceptions.** Java distinguishes between two types of exceptions—checked and unchecked. C# chose a more minimalist approach by only having one type of exception. While the ability to catch exceptions can be useful, it can also have an adverse effect on scalability and version control.
- **Polymorphism:** C# and Java take very different approaches to polymorphism. While Java enables polymorphism by default, C# must invoke the "virtual" keyword in a base class and the "override" keyword in a derived class.
- **Enumerations (Enums):** In C#, enums are simple lists of named constants where the underlying type must be integral. Java takes the enum further by treating it as a named instance of a type, making it easier to add custom behavior to individual enums.

When Should You Use C# or Java?

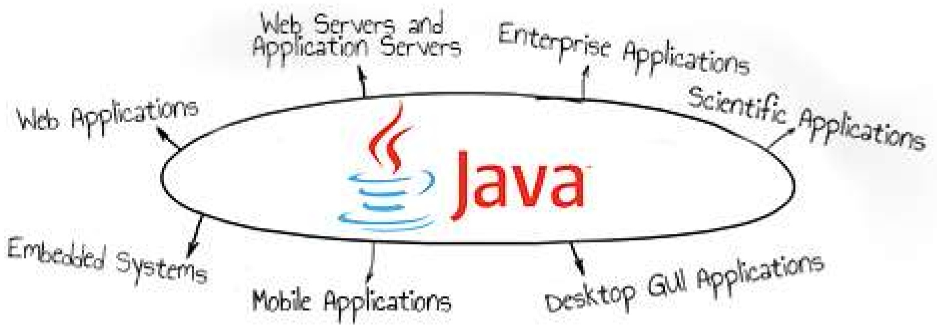
With so much in common, the language you ultimately choose to use will depend largely upon the platform you have chosen for your project. Today, C# is used primarily on the .NET Framework, Mono, and Portable.NET implementations of the CLI. If your software or web application is being built for Windows, C# will work best with the .NET suite of technologies.

That said, if you wish to develop for Unix, Linux, or other platforms outside of the Microsoft platform, Java's large open-source ecosystem is the better choice. The community is constantly creating new libraries and tools. New powerful languages like Scala, Clojure, and Groovy have also appeared, all based off of the JVM. It also doesn't hurt that most JVM implementations are open-source and free. Java is also the main language used by Google to develop for Android, which is currently the largest mobile operating system in the world.

Keep in mind that the advantages listed above are slight, and neither language is likely to disappear anytime soon. Both languages have been around long enough that there's not really anything you can't build in one that you couldn't build in the other. *Bottom line:* Choose the language that works best for your project's platform of choice.

Applications of Java Programming Language

By Arvind Rongala, Manager, Business Development and Marketing, Invensis Technologies



Java is widely applicable across different types of applications

Java language was developed by Sun Microsystems in 1995. In subsequent years, the language has become the backbone of millions of applications across multiple platforms including Windows, Macintosh and UNIX-based desktops, Android-based mobiles, embedded systems and enterprise solutions. According to Oracle (that acquired Sun Microsystems in 2010), Java now runs on more than 3 billion devices.

Types of Applications that Run on Java

1. Desktop GUI Applications:

Java provides GUI development through various means like Abstract Windowing Toolkit (AWT), Swing and JavaFX. While AWT contains a number of pre-constructed components such as menu, button, list, and numerous third-party components, Swing, a GUI widget toolkit, additionally provides certain advanced components like trees, tables, scroll panes, tabbed panel and lists. JavaFX, a set of graphics and media packages, provides Swing interoperability, 3D graphic features and self-contained deployment model which facilitates quick scripting of Java applets and applications.

2. Mobile Applications:

Java Platform, Micro Edition (Java ME or J2ME) is a cross-platform framework to build applications that run across all Java supported devices, including feature phones and smart phones. Further, applications for Android, one of the most popular mobile operating systems, are usually scripted in Java using the Android Software Development Kit (SDK) or other environments.

3. Embedded Systems:

Embedded systems, ranging from tiny chips to specialized computers, are components of larger electromechanical systems performing dedicated tasks. Several devices, such as SIM cards, blue-ray disk players, utility meters and televisions, use embedded Java technologies. According to Oracle, 100% of Blu-ray Disc Players and 125 million TV devices employ Java.

4. Web Applications:

Java provides support for web applications through Servlets, Struts or JSPs. The easy programming and higher security offered by the programming language has allowed a large number of government applications for health, social security, education and insurance to be based on Java. Java also finds application in development of eCommerce web applications using open-source eCommerce platforms, such as Broadleaf.

5. Web Servers and Application Servers:

The Java ecosystem today contains multiple Java web servers and application servers. While Apache Tomcat, Simple, Jo!, Rimfaxe Web Server (RWS) and Project Jigsaw dominate the web server space, WebLogic, WebSphere, and Jboss EAP dominate commercial application server space.

6. Enterprise Applications:

Java Enterprise Edition (Java EE) is a popular platform that provides API and runtime environment for scripting and running enterprise software, including network applications and web-services. Oracle claims Java is running in 97% of enterprise computers. The higher performance guarantee and faster computing in Java has resulted in high frequency trading systems like Murex to be scripted in the language. It is also the backbone for a variety of banking applications which have Java running from front user end to back server end.

7. Scientific Applications:

Java is the choice of many software developers for writing applications involving scientific calculations and mathematical operations. These programs are generally considered to be fast and secure, have a higher degree of portability and low maintenance. Applications like MATLAB use Java both for interacting user interface and as part of the core system.

In conclusion, Java is widely applicable across different types of applications. It offers cross-functionality and portability, and these features, among many others, make Java the programming language of choice for software development of a specific nature.

C# and Java Keyword Comparison

Comparing the keywords in C# and Java gives insight into major differences in the languages, from an application developer's perspective. Language-neutral terminology will be used, if possible, for fairness.

Equivalents

The following table contains C# and Java keywords with different names that are so similar in functionality and meaning that they may be subjectively called "equivalent." Keywords that have the same name and similar or exact same meaning will not be discussed, due to the large size of that list. The Notes column quickly describes use and meaning, while the example columns give C# and Java code samples in an attempt to provide clarity.

It should be noted that some keywords are context sensitive. For example, the *new* keyword in C# has different meanings that depend on where it is applied. It is not used only as a prefix operator creating a new object on the heap, but is also used as a method modifier in some situations in C#. Also, some of the words listed are not truly keywords as they have not been reserved, or may actually be operators, for comparison. One non-reserved "keyword" in C# is *get*, as an example of the former. *extends* is a keyword in Java, where C# uses a ':' character instead, like C++, as an example of the latter.

C# Keyword	Java Keyword	Notes	C# Example	Java Example

Base	super	Prefix operator that references the closest base class when used inside of a class's method or property accessor. Used to call a super's constructor or other method.	`public MyClass(string s) : `**`base`**`(s)`
			`{`
			`}`
			`public MyClass() : `**`base`**`()`
			`{`
			`}`
				`Public MyClass(String s)`
				`{`
				`super``(s);`
				`}`
				`public MyClass()`
				`{`
				`super``();`
				`}`

Bool	boolean	Primitive type which can hold either true or false value but not both.	**`bool`**` b = true;`	**`boolean`**` b = true;`

Is	instanceof	Boolean binary operator that accepts an l-value of an expression and an r-value of the fully qualified name of a type. Returns true iff l-value is castable to r-value.	`MyClass myClass = new MyClass();`
			`if (myClass `**`is`**` MyClass)`
			`{`
			`//executed`
			`}`
				`MyClass myClass = new MyClass();`
				`if (myClass `**`instanceof`**` MyClass)`
				`{`
				`//executed`
				`}`

lock	synchronized	Defines a mutex-type statement that locks an expression (usually an object) at the beginning of the statement block, and releases it at the end. (In Java, it is also used as an instance or static method modifier, which signals to the compiler that the instance or shared class mutex should be locked at function entrance and released at function exit, respectively.)	`MyClass myClass = new MyClass();` `lock (myClass)` `{` `//myClass is` `//locked` `}` `//myClass is` `//unlocked`	`MyClass myClass = new MyClass();` `synchronized (myClass)` `{` `//myClass is` `//locked` `}` `//myClass is` `//unlocked`
namespace	package	Create scope to avoid name collisions, group like classes, and so on.	**`namespace`** `MySpace` `{` `}`	`//package must be first keyword in class file` **`package`** `MySpace;` `public class MyClass` `{` `}`
readonly	const	Identifier modifier allowing only read access on an identifier variable after creation and initialization. An attempt to modify a variable afterwards will generate a compile-time error.	`//legal initialization` **`readonly`** `int constInt = 5;` `//illegal attempt to` `//side-effect variable` `constInt = 6;`	`//legal initialization` **`const`** `int constInt = 5;` `//illegal attempt to` `//side-effect variable` `constInt = 6;`
sealed	final	Used as a class modifier, meaning that the class cannot be subclassed. In Java, a method can also be declared final, which means that a subclass cannot override the behavior.	`//legal definition` `public` **`sealed`** `class A` `{` `}` `//illegal attempt to` `//subclass - A is`	`//legal definition` `public` **`final`** `class A` `{` `}` `//illegal attempt to` `//subclass - A is`

			//sealed public class B: A { }	//sealed public class B extends A { }
using	import	Both used for including other libraries into a project.	**using** System;	**import** System;
internal	private	Used as a class modifier to limit the class's use inside the current library. If another library imports this library and then attempts to create an instance or use this class, a compile-time error will occur.	namespace Hidden { **internal** class A { } } //another library using Hidden; //attempt to illegally //use a Hidden class A a = new A();	package Hidden; **private** class A { } //another library import Hidden; //attempt to illegally //use a Hidden class A a = new A();
:	extends	Operator or modifier in a class definition that implies that this class is a subclass of a comma-delimited list of classes (and interfaces in C#) to the right. The meaning in C# is very similar to C++.	//A is a subclass of //B public class A : B { }	//A is a subclass of //B public class A **extends** B { }
:	implements	Operator or modifier in a class definition that implies that this class implements a comma-delimited list of interfaces (and classes in C#) to the right. The meaning in C# is very similar to C++.	//A implements I public class A : I { }	//A implements I public class A **implements** I { }