# Can C# Replace Java in CS1 and CS2?

Stuart Reges
Department of Computer Science
University of Arizona
Tucson, AZ 85712
reges@cs.arizona.edu

## ABSTRACT

Microsoft has developed a language called C# ("see sharp") that it claims will allow programmers to "quickly and easily build solutions" for its new.NET platform [3]. The language has much in common with Java, particularly in those features emphasized in CS1 and CS2 courses. It also includes some of the desirable features of C++ that are missing from Java as well as some new features not available in either language. This paper explores the pros and cons of teaching CS1 and CS2 using C# instead of Java and concludes with a discussion of the author's plans for teaching such a course in the fall of 2002.

**Categories & Subject Descriptors:** D.3.3 **[Programming Languages**]: Language Contructs and Features – classes and objects, control structures, data types and structures, dynamic storage management.

**General Terms:** Languages.

**Keywords:** CS1, CS2, object oriented programming, C#, Java.

## 1. Motivation

Bill Gates has referred to the .NET technologies as "the breakthrough tools that will allow developers to write the next generation of applications" [2]. He might be spouting marketing hype, but the release of .NET clearly constitutes a watershed for the company.

Many schools will choose to ignore what Microsoft does, but others will find the Microsoft connection appealing. Few departments would sacrifice the quality of instruction just to teach .practical. tools that are currently used in industry. But if C# turns out to be as useful as Java in teaching the CS curriculum, some schools will choose C# instead of Java because of the particular interests of their students, their school or their community.

There has been a shift away from C++ in computer science education in recent years. The reasons for this shift were documented in a report from an ad-hoc committee formed by the College Board to plan for the future of the AP/CS course[1]. They mention that their primary reasons for changing were type safety, simplicity and object orientation. They felt that Java was superior to C++ in all three aspects. These goals were also primary in the design of C#. The C# language specification says that, "C# is a simple, modern, object oriented, and type-safe programming language derived from C and C++ [4]."


*ITiCSE'02,* June 24-26, 2002, Aarhus, Denmark.


Schools that have already switched to Java are unlikely to want to switch again, but many schools are still using C++ and they might decide that switching to C# instead of Java makes sense for them.

## 2. The Overlap of C# and Java

Microsoft characterizes C# as a variant of C++, but it most closely resembles Java.

The two have many common characteristics.

- automatic garbage collection
- type safety
- no free functions/variables (everything in a class)
- runtime checks on type cast
- single inheritance
- interfaces, multiple implementation of interfaces
- reflection
- exceptions with try/catch
- threads
- a rich set of collection classes

Consider the following short Java program.

```
public class Figure
{
  public static void main(String[] args)
  {
    for (int i = 0; i < 6; i++) {
      for (int j = 0; j < 6; j++)
        if (i == j)
          System.out.print("\\");
        else if (i == 6 - j - 1)
          System.out.print("/");
        else
          System.out.print("*");
      System.out.println();
    }
  }
}
```

This produces a simple "X" figure as output:

```
\****/
*\**/*
**\/**
**/\**
*/**\*
/****\
```

The corresponding C# program is nearly identical:

```
using System;

public class Figure
{
  public static void Main(string[] args)
  {
    for (int i = 0; i < 6; i++) {
      for (int j = 0; j < 6; j++)
        if (i == j)
```

```
          Console.Write("\\");
        else if (i == 6 - j - 1)
          Console.Write("/");
        else
          Console.Write("*");
      Console.WriteLine();
    }
  }
}
```

As in Java, methods associated with a class are declared static and a special static method called Main launches each application. The control structures are nearly identical between the two languages. And in place of Java's System.out with print and println methods C# has Console.Write and Console.WriteLine methods.

Even class declarations are nearly identical. Consider the following Java class.

```
public class Point
{
  public Point(double x, double y)
  {
    myX = x;
    myY = y;
  }

  public Point()
  {
    this(0.0, 0.0);
  }

  public String toString()
  {
    return "(" + myX + ", " + myY + ")";
  }

  private double myX;
  private double myY;
}
```

And consider its C# counterpart.

```
public class Point
{
  public Point(double x, double y)
  {
    myX = x;
    myY = y;
  }

  public Point(): this(0.0, 0.0)
  { }

  public override string ToString()
  {
    return "(" + myX + ", " + myY + ")";
  }

  private double myX;
  private double myY;
}
```

As in Java, string concatenation is used often and the language has a standard .to string. method in the object class that is overridden in subclasses. Constructors and class methods are defined with a similar syntax, although C# requires explicit overriding of the inherited ToString method.

C# is not, however, just another implementation of Java. It has many features that Java does not, including enums, reference parameters, structs, properties, indexed properties and operator overloading. These extra features increase the overall complexity of the language, but an instructor could teach a Java-like CS1 and CS2 course without covering these. In that sense, they provide more of an opportunity than an incumbrence. Many instructors, for example, will find it refreshing to be able to write a true swap function, which can't be done with the value parameter mechanism in Java.

## 3. The Benefits of C# over Java

In 1997 Mark Weiss presented the first significant SIGCSE paper describing the use of Java in CS2[6]. He warned us in a section entitled "Java Annoyances" that Java is by no means perfect. Many of us have since come to learn about those annoyances from firsthand experience.

Eric Roberts provided a long list of Java annoyances in describing his "MiniJava" language intended for novices [5]. Most of the simplifications he has included in defining MiniJava have been included in C# as well, although C# can by no means be considered the kind of "mini" language he advocates for novices.

C# is also by no means perfect, but the language designers had the benefit of knowing about Java's weaknesses and this allowed them to address many of them.

### 3.1 Standard Input

It is fairly easy to produce simple console output in Java, but reading simple console input is anything but simple. One would think that if System.out is the object to use for writing to the console that System.in would be the object for reading from the console. But System.in has a confusing set of reading operations (read one byte as an int or read into a byte array) and they all potentially throw an IO exception that must be handled.

Of course, an experienced Java programmer will know how to read from the console, but because the standard class libraries do not make it easy, everyone comes up with their own solution. There are many different "easy input" classes that are available, but every textbook seems to have its own class. Some textbooks stay with standard Java and show students how to construct a BufferedReader object, but then one is stuck with the problem of exceptions. Most authors are wise enough not to try to cover try/catch blocks just to read console input, but then they end up advising students to say "throws IOException" at the end of the main method and any other method that performs reading operations. This is confusing for beginners and encourages bad habits.

As demonstrated in the sample program in the previous section, C# provides the standard utilities Console.Write and Console.WriteLine that are exact parallels of Java's System.out.print and System.out.println. C# also provides Console.Read for reading one character of input and Console.Readline for reading an entire line of input. They are simple to use, as in the example below.

```
Console.Write("What is your name? ");
string name = Console.ReadLine();
```

Reading numbers is a bit more work, but not nearly as difficult as in Java.

```
Console.Write("Give me an int-> ");
int x = int.Parse(Console.ReadLine());
```

## 3.2 Simpler Main

The sample C# program in the previous section had a main method with a signature almost identical to the standard Java main method. But in C#, as in C and C++, one can use a simpler Main method. For example, the classic hello world program can be written as follows in C#.

```
using System;

public class Hello
{
  public static void.Main()
 {
    Console.WriteLine("hello world");
  }
}
```

It may seem like a trivial issue to be able to leave out the "string[] args" parameter in the main method, but this is the kind of confusing detail that makes Java a poor choice for novice programmers. We find ourselves having to tell our students to "just ignore that" or that "we'll get to that eventually" and for now to just "think of that as a magical incantation that you always have to include."

## 3.3 Consistent Object Model

Java makes a distinction between primitive types like int, double, char and boolean and reference types (objects). To use a primitive value like an int where Java expects an object, one must construct a "wrapper" object that holds the primitive data. One must also unwrap the value when trying to get it back. For example, pushing two ints onto a stack and then popping them off again requires the following code.

```
Stack s = new Stack();
s.push(new Integer(19));
s.push(new Integer(28));
int x = ((Integer)s.pop()).intValue();
int y = ((Integer)s.pop()).intValue();
```

This is difficult for novices to understand and creates an artificial distinction C# has a more consistent object model. Every type, even types like int and double, are treated as objects. One can call the ToString method of a simple int, just as one can for any other object. And one can use ints wherever one would use an object. The stack code would be written as follows in C#.

```
Stack s = new Stack();
s.push(19);
s.push(28);
int x = (int)s.pop();
int y = (int)s.pop();
```

There is still a cast when removing an element from the stack, but that's because the stack is defined in terms of the generic type object. The cast is exactly what you'd expect it to be—to type int. If we'd pushed complex objects onto the stack, we would have needed the same kind of cast when removing values.

C# provides this more consistent object model by automatically "boxing" and "unboxing" simple data like ints. In effect, wrapper objects are used when necessary, but the compiler does all of the work, not the programmer.

One of the primary benefits of this more consistent object model is that CS1 courses can explore the use of collections classes much earlier than they would otherwise. Most Java CS1 books cover arrays before they discuss structures like ArrayList because it isn't easy to manipulate simple values like ints in an Arraylist. C# has no such limitation, so it would make sense to have students using ArrayList and other collections early in the course before they learn how to implement such structures themselves.

## 3.4 Iterators and the foreach Loop

The iterator pattern is one of the most fundamental in object-oriented programming, yet it is considered by most authors to be an advanced topic. Students first learn "simpler" loops like array-processing loops before they study iterators. There is nothing intrinsically complex about iterators. In fact, the concept is much simpler than that of array indexing.

C# has the same for loops and while loops that Java has, but it also has a simple foreach loop that can be used for any collection that has an iterator (i.e., that implements a standard interface known as IEnumerable). For example, the following code constructs an array list, adds various integers to it and then computes the sum of the values.

```
ArrayList list = new ArrayList();
list.Add(8);
list.Add(7);
list.Add(19);
list.Add(308);

int sum = 0;
foreach (int x in list)
  sum += x;
Console.WriteLine("sum = " + sum);
```

This kind of code could be discussed early in a CS1 course. The syntax of the foreach loop is so simple that it might be desirable to teach it before teaching for loops and while loops.

The existence of the foreach loop also provides an incentive to the students to include an iterator in any structure that they define. Any C# programmer will be familiar with the foreach loop and the iterator will provide a simple and convenient way to access the elements. Thus, instead of being an advanced topic, iterators become an early topic and an idea that is integrated into the language itself.

## 3.5 Properties

Novices find it at least annoying and often confusing to have to define and call get and set methods. Suppose, for example, that you are defining a point class for storing x and y coordinates and you wanted to set one point's coordinates to twice another point's coordinates. In Java you'd say something like the following:

```
p2.setX(p1.getX() * 2);
p2.setY(p1.getY() * 2);
```

Of course, one could use public data fields to simplify the syntax, but then one loses encapsulation. C# allows a programmer to have the simplified syntax and preserve encapsulation by defining what is known as a property. A property can be accessed using the same syntax as a data field, but the access is translated by the compiler into calls on get and set methods defined in the property. So the point manipulating code above with C# properties becomes:

```
p2.X = p1.X * 2;
p2.Y = p1.Y * 2;
```

These lines of code are much simpler for a novice to understand. Properties become particularly handy when the operations are

even more complex or involve familiar concepts like incrementing with the ++ operator, as in:

```
p2.X = (p1.X + p1.Y)/(p2.X + p2.Y);
p2.Y++;
```

Once the get and set methods for the property are defined, the client of the code does not have to figure out which one to call. The compiler does this for the client, which makes life much simpler.

## 3.6 Convenient Custom Classes

No matter how rich the class libraries of a language are, most instructors will want to provide some custom classes for their students to use. C# provides convenient features for simplifying the interface for such classes. As described in the previous section, properties can make client code easier to write. C# also allows operator overloading and type conversions. While an instructor might choose not to teach students how to do operator overloading or how to define conversions, the instructor will find it useful to use these tools in creating classes for their students to use. This will allow them to create the simplest possible client interfaces for their students.

For example, when I taught CS1 in C++ I used a custom class written by Owen Astrachan of Duke University for manipulating dates. It was fairly simple for students to figure out things like how many days old they are. This seemed like a much more interesting program to write than a program that adds two numbers together. A person can add the numbers by themselves easily, but figuring out how many days you've been alive isn't trivial.

I stopped using this assignment in Java because the standard date classes are too complex to use. In C# the assignment again is reasonable. Using standard classes, students can write the following code.

```
DateTime bday = new DateTime(1983, 9, 6);
TimeSpan age = DateTime.Now - bday;
Console.WriteLine(("I am :");
Console.WriteLine(age.TotalDays + " days old");
Console.WriteLine(age.TotalMinutes +
                  " minutes old");
Console.WriteLine(age.TotalSeconds +
                  " seconds old");
```

It is particularly convenient that the classes provided by the standard C# libraries are so easy to use, but even if they weren't, the language provides the ability to define your own custom class with the behavior you prefer.

Another useful feature for creating custom classes is the ability to define indexed properties. This is similar to overloading the [] operator in C++. If you have designed a collection, you can allow people to use an array-like syntax to refer to individual elements.

## 3.7 Miscellaneous

C# includes several other nice features that have the potential of improving a CS1 or CS2 course. Novices often accidentally leave off a break statement in defining a switch statement. This is not allowed in C# and will lead to a helpful compiler error message rather than a subtle bug. C# includes reference parameters, so that on those occasions when you really want to write a swap function, you can. It includes enumerated types that add to readability. One can even write out an enumerated type and C# writes out the name of the constant, not an int. C# also has what are known as "delegates." A delegate can be thought of as a type-safe function pointer. This might lead to interesting possibilities particularly in the CS2 course. For example, one might define a "map" or "forall" method that takes a function and applies it to every element of the collection. Or one might provide a boolean function to filter a list. The C++ Standard Template Library provides many such utilities, but the syntax and semantics of STL iterators are rather complex for CS2 students.

# 4. What about the JVM?

Another distinguishing characteristic of Java is that it compiles not to native code but to Java .bytecodes. to be executed on the Java Virtual Machine (JVM). This intermediate language makes it easier to execute Java code across many platforms. You simply need a JVM for each platform you want to use.

While this is certainly an interesting aspect of Java, we don't tend to teach much about it to our students in CS1 and CS2. We mention it, but few students really understand this distinction. So it's not clear that we would lose much by switching to C#.

C# also has its own interesting properties. Microsoft does not use the term "virtual machine" in describing the C# compiler, but there is a C# virtual machine. The language is compiled into an intermediate language known as MSIL (Microsoft Intermediate Language). This intermediate language is very similar to Java bytecodes. Microsoft does not call this a virtual machine because unlike Java, which is interpreted, MSIL is just-in-time compiled to native code before execution.

# 5. What About MultiPlatform?

The single biggest difference between Java and C# is that Java was designed to run on multiple platforms and C# was designed to run with multiple languages. Some people have claimed that it is important to teach the concept of multiple platforms to our students, but this is really more of a logistical question than a pedagogical question. If you use Java in your CS1 and CS2 courses, your students will have a number of platforms to choose from for completing their assignments (Windows, Unix, linux, MacOS, etc). If you use C# in your CS1 and CS2 courses your students will be limited (at least for now) to Windows as a platform.

Again, I don.t believe that we teach our students how Java manages to be multiplatform. Those who do will find that they can substitute a discussion of how C# manages to work well with other languages. The .NET platform includes a Common Language Runtime (CLR) that allows one to mix code from multiple languages. All of the code compiles to the same intermediate language, so it is extremely interchangeable. For example, one can take code written in C++ and extend it in C#, writing C# classes that inherit from C++ classes.

# 6. But isn't Java .Cool.?

As Mark Weiss pointed out in his 1997 paper, .large numbers of students were excited to be learning cutting-edge technology— "Although many students indicated that they worked harder in this course than any other, most seemed eager to do so, because they felt they were learning a marketable skill [6]."

Java may still have an aura of .cool,. but it is no longer cutting-edge. Sun and others have worked tirelessly to extend the class

libraries to allow Java programmers to keep up with technology trends, but eventually these extensions upon extensions upon extensions start to become incomprehensible.

This is another area where Microsoft had the benefit of coming to the party late. They had the opportunity to incorporate the latest technologies like XML and SOAP into their class libraries and their development tools. Microsoft claims that programmers will find it much easier to create web services and to program mobile devices using the new .NET framework. The jury is still out on whether or not that's true, but Microsoft had a wonderful opportunity to start from scratch rather than extending an existing framework and it's likely that they got a lot of it right.

## 7. Future Plans

After teaching C# to a group of junior/senior level undergraduates, I have come to believe that C# might work well as a CS1/CS2 language. I hope to offer a C# course in fall 2002 for honors students. Our department offers an accelerated one-semester course that covers all of CS1 and CS2 in 15 weeks. I hope to cover the same material in 13 weeks using C# so that I'll have two weeks to discuss how Java differs from C# and to have the students write some Java programs. That way they should leave the course with the ability to program in either language. This goal would probably not be attainable with mainstream students, but it should be possible with a small group of honors students.

## 8. Conclusions

C# has tremendous overlap with Java, which means that as a language it will probably be as effective as Java in teaching CS1 and CS2. It fixes many of the shortcomings of Java that have been particularly difficult for novice programmers and it provides extra features that have the potential to enrich the course.

C# is, however, more complex than Java and there is always a price for complexity. Every extra keyword, construct and control structure makes the language more difficult for a novice. One can teach a subset, but students are often dissatisfied with this approach.

On the whole, the languages are probably comparable. Some will argue that Java's six of one is greater than C#'s half dozen of another and others will argue the opposite, but there is far more overlap than difference, particularly for the features most often taught in CS1 and CS2.

Although this paper has mostly ignored political and logistical questions, those are likely to be the issues that determine how individual schools proceed.

## 9. References


[1] Astrachan, et al. Recommendations of the AP Computer Science Ad Hoc Committee, 2000. http://www.collegeboard.org/ap/computer-science

[2] Gates, Bill, speech to TechEd 2001, http://www.microsoft.com/billgates/speeches/2001/06-19teched.asp

[3] MSDN, C# Introduction and Overview, 6/26/00, http://www.msdn.microsoft.com/vstudio/nextgen/technology/csharpintro.asp

[4] MSDN, C# Language Specification, Version 0.28, 5/7/01, http://msdn.microsoft.com/vstudio/nextgen/technology/Csharp_Language_Specification.doc

[5] Roberts, Eric, An Overview of MiniJava, SIGCSE Symposium 2001, pages 1-5

[6] Weiss, Mark, Experiences Teaching Data Structures with Java, SIGCSE Symposium 1997, pages 164-168