IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Code Conversion From Java to C#

[1]Vipul Ghadge, [2]Satyam Sabale, [3]Vinit Pakhale, [4]Devanjan Mukherjee, [5]Mrs. Dhanashree Phalke, [6]Mr. Jayant Shimpi

[1-4]Student, [5-6]Assistant Professor,

Department of Computer Engineering, D.Y.Patil College of Engineering Akurdi, Pune, India

Abstract: Nowadays there are lots of programming languages in use widely, only some of them appear as widely used and learned. Many of them require a calculation sequence (imperative form); others simply indicate the desired results (declarative form). System proposed some ways to convert java code to C# in this paper. The proposed algorithm which works for the proposed code conversion approach is called Source Code Migration (SCM) algorithm. After implementing this algorithm it ensures that algorithm works for code conversion from java to C# language.

***Keywords* - Java, C#, programming language, conversion**.

I. INTRODUCTION

The indistinguishable code is written in so many languages. Therefore, thousands of systems have existed and more are made, although only a small fraction of the programming languages appear as commonly used and understood. Mostly of them need a series of measurements (imperative form); others simply state the desired results (declarative form). In this paper, proposed system is aims to find some ways of translating java code to C#. Although Java is a young language, it rapidly produced legacy Java programs due to the rapid development of Java requirements. The introduction of C# and .Net created a demand for tools and techniques to turn legacy applications into the new language and interface, as with many other innovations. System has created an experimental converter which is partially automatic to convert a subset of Java 1.1 into C#. Partial here has two significance. First and foremost, it only consider a specific chunk of the Java language, as it was designed as an exercise not for full commercial implementation. Secondly, for certain language constructs, the automatic conversion is almost impossible, and some manual transformation is required. For those later cases, the transformer logs the problem and leaves notes for the developer to finish the transition manually in the converted source code.

II. LITERATURE SURVEY

Some common and some popular analysis and commercial language translation methods will tackle certain S2ST problems with differing success rates. The larger the semantic a syntactic gap between the languages of source and target, the harder the conversion is [2]. Microsoft Java Language Conversion Assistant (JLCA) is the nearest function to Java to C#, currently available in version 2.0 and Beta version 3.0 [1]. As proprietors say, it can transform Java applications to J++ and C# and Java API calls to native. Net Framework calls, 80 percent automatically. For a discussion of JLCA, an example of code conversion, some of the issues that arise during API conversion, and the manual coding needed to complete the conversion task, see [3]. What conversion technology is being used in JLCA is unknown, since it is proprietary. It's also unclear how the JLCA is flexible and extendable. Java to C# is unusual in using rewrite of a parse tree by means of a by-example rule specification form, as in TXL. Extensions to cover other versions of Java, adding new transformation rules or changing existing ones are well supported by the transformer's flexibility [5]. Java to C# is special in being mindful of the extensive and subtle variations between the two languages. For example, Java to C# is watchful of and able to transform the default access (when no access modifiers are defined in the code). Access to Java by default is Nice, while access to C# by default is private. Ignoring those settings in the resulting code will be troublesome [4].

SR. NO	**PAPER DETAIL**	**METHODS**	**RESULT**
1.	W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba, "Taxonomy of cross-platform mobile applications development approaches , 2016	Compilation approach	two sub-approaches: Cross-Compiler And Trans-Compiler.
2.	H. Zhong, S. Thummalapenta, T. Xie, L. Zhang, and Q. Wang, "Mining Api Mapping for Language Migration, 2010"	Trans-Compiler Approach	The code conversion is a nontrivial task because mapping the API between two languages is difficult
3.	P. Klima and S. Selinger, "Towards Platform Independence of Mobile Applications,2013	proposed to convert the Android Apps to platform independent web Apps by using the Google Web Toolkit (GWT)	Features in the Android source system which are neither supported by GWT nor by HTML5 are ignored.
4.	*J2ObjC*. Available: https://code.google.com/p/j2objc/ [Last Visited: 22/3/2015]	Proposed J2ObjC tool	It supports the transformation Of general business logic as the transformation of the UI-specific code is not supported.
5.	*JUniversal*. Available: http://juniversal.org/ [Last Visited: 22/3/2015]	Proposed JUniversal tool which converts the Java source code files of the Android platform to the equivalent C#	Universal doesn't provide any support for UI today and the developer has to write it natively.

III. MODEL DRIVEN DEVELOPMENT APPROACH

Using this method, application-specific versions of the App are created from a platform independent model. MD2 is a solution that makes use of that strategy. MD2 framework is based on a new DSL adapted to the Mobile Apps domain. This framework allows applications to be created by defining the appliance model using the newly defined DSL, then a series of integration steps to supply different source code or program for the native and platforms.

IV. RUNTIME INTERPRETATION APPROACH

Runtime is an environment for execution, and a layer that makes the mobile app run on the native platform. This method aims to convert the target code into bytecode, which then is executed at runtime by a virtual machine assisted by the mobile device. Titanium is an appcelerator which is free and open source application development platform.

V. CROSS-COMPILER APPROACH

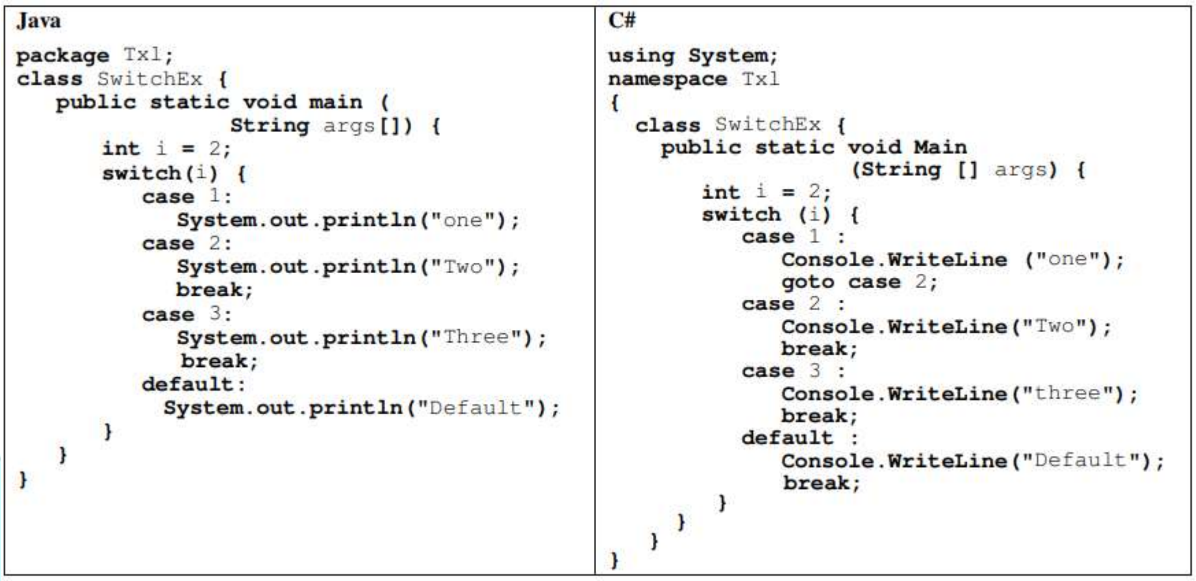
A cross-compiler may be a compiler running on a computer that features a different OS than the one running the compiled program on. XMLVM which uses this approach can be considered as a solution. The XMLVM tool is a cross-compiler of language type bytes.
XMLVM cross-compiles the byte code instructions from the virtual machine of Sun Microsystem instead of cross-compiling at the source code level. XMLVM not only cross-compiled language-level applications but also maps the APIs between different platforms. The advantages of choosing the byte-code transformation include:

1) Byte codes are much harder to decipher than Java source code;
2) Certain high-level language features such as generics are already reduced to low-level byte code instructions; and
3) The Java compiler does thorough modifications to generate effective byte codes.

Nonetheless, it is very difficult to get mapping between the source language and the target language. The cross-compiler also follows a few platforms and only relies on the common elements of those platforms.
Consumer price level (CPI) is employed as a proxy during this study for rate of inflation. CPI may be a wide basic measure to compute usual variation in prices of products and services throughout a specific period of time. It is assumed that arise in inflation is inversely associated to security prices because Inflation is at last turned into nominal interest rate and change in nominal interest rates caused change in discount rate, so rate of discount increase on account of increase in rate of inflation and increase in discount rate leads to decrease the cash flow's present value (Jecheche, 2010). The purchasing power of money decreased due to inflation, and due to which the investors demanded a high rate of return, and the prices decreased with increase in required rate of return (Iqbal et al, 2010).

VI. Trans-Compiler Approach

Transforming one high-level programming language into another high-level programming language is done using a trans-compiler. In general, the translation of code is a non-trivial task, as mapping the API between two languages is a complex task for the following reasons: 1) class names are different, 2) function names are way different, and 3) the number and types of function parameters are also different. The use of the Google Web Toolkit (GWT) is to transform Android Apps to platform-independent Web Apps is recommended. The converter uses the Java Development Tools (JDT) from Eclipse to manipulate the source code, because both Android and the target GWT framework have their own custom Java library (Java runtime library subset). The converter's main aim is to deal with the variations in those two Java libraries. The converter needs to identify all source code statements which are not supported in the target system. The source code for Android is being parsed to generate AST. If the converter finds library calls that are not provided by the GWT (Java-to-JavaScript compiler), this will change the syntax tree.



Java

```java
package Txl;
class SwitchEx {
   public static void main (
                String args[]) {
      int i = 2;
      switch(i) {
         case 1:
            System.out.println("one");
         case 2:
            System.out.println("Two");
            break;
         case 3:
            System.out.println("Three");
             break;
         default:
           System.out.println("Default");
      }
   }
}
```

C#

```csharp
using System;
namespace Txl
{
  class SwitchEx {
    public static void Main
                (String [] args) {
       int i = 2;
       switch (i) {
          case 1 :
             Console.WriteLine ("one");
             goto case 2;
          case 2 :
             Console.WriteLine("Two");
             break;
          case 3 :
             Console.WriteLine("three");
             break;
          default :
             Console.WriteLine("Default");
             break;
       }
    }
  }
}
```

Figure 1. Example of conversion of program

VII. Proposed Method

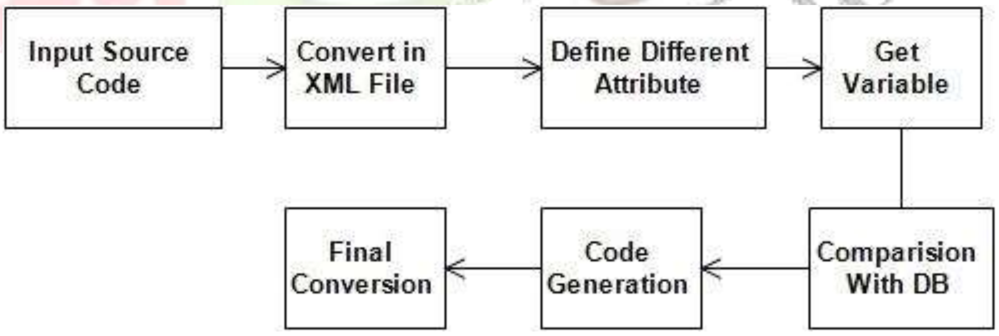


Figure 2: System Architecture

The proposed algorithm that applies the source code conversion approach is called Source Code Migration (SCM) algorithm. The main aim of the SCM algorithm is to search in the input source code for toning a predefined set of code patterns, and then get the equivalent source code for the target platform from the ICPMD database.

Input Source Code

The input to the proposed system is a source code file that is written in the source platform programming language.

Convert in XML File

This input is converted to a transitional XML representation file, and then the XML file is used to generate the equivalent source code file which is written in the programming language of the target platform

Define Different Attribute

In the proposed system there must be defined all attributes of both programming languages so that it will easily be converted to a secondary language. Some of them are user defined and the others are system defined attributes.

Get Variable

After the declaration of the attribute, the proposed system has to look for how many variables are there and how to define it in another language. So that the next course can be done.

***Code Preparation**:*

This stage parses all the lines of code in the input code file to delete any ambiguities that could hinder the detection of the code patterns. In most cases, two or more types of code statements could exist on the same line of source code which require to be separated into different lines: Source code statement and comment statement on the same line.

***Code Conversion**:*

This stage parses and translates all the source code lines, in the updated source code file, to the intermediate abstract model XML representation which contains translated code for all target platforms. The main idea is to compare each line in the input source code with the predefined patterns, then compose the output XML representation file as follows:

If a line is matched with one of the code patterns: Add a new element <comment> contains the recognized source code line.

Code Generation:

This stage uses the language XSLT file to convert the intermediate XML representation file to the source code file of the target platform.

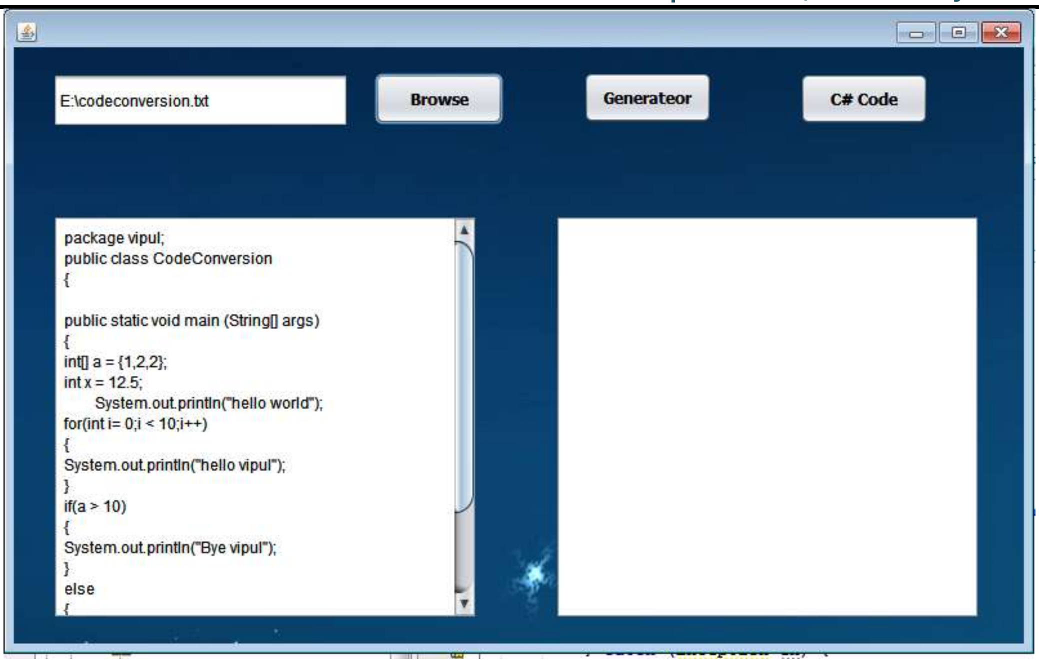
VIII. Transforming small program from Java to C#

Java to C# has been tested on a number of small size applications. The figure 1 above is the good example of how a utter transformation of the program is done, using some of the above and other rules. It is because of space barrier, it's a very small example. Using Java to C# rules and functions at the end of the example, system required description and specifics of how the transition occurs. The key transition law fits the entire Java system, consisting of a package header, an import declaration, and a description statement. The package header is redefined to include the notation C#, and the function changePackageToNamespace transforms it into a declaration of namespace C#. Import declaration is voluntary, and there is no import clause in the program. The default namespace of the source code in C# is automatically concatenated to the software because it includes the bulk of the required API and utility classes for basic programs.

The proposed system contains the statement of one type that is the class all alone. ChangeClassHeader matches the class statement and is used to treat the class as a whole. This splits the class down to its header and frame. Upon calling changeModifiers, changeImplement, and changeExtend a new class header is created to do more advanced research. TranslateEmptyBody transforms a class body for an empty body, and changesClassBody for body with fields, instance initializers, static initializers, constructors, and statements of method. The class body contains a method statement matched with translateMethods. The pattern match for the function is the main special method. This consists of modifiers, type specifier, an optional clause on throws and method body as a method. The main awesome thing about the' main' Java approach is that it turns the name into' Main' with capital M in C#.

IX. Results

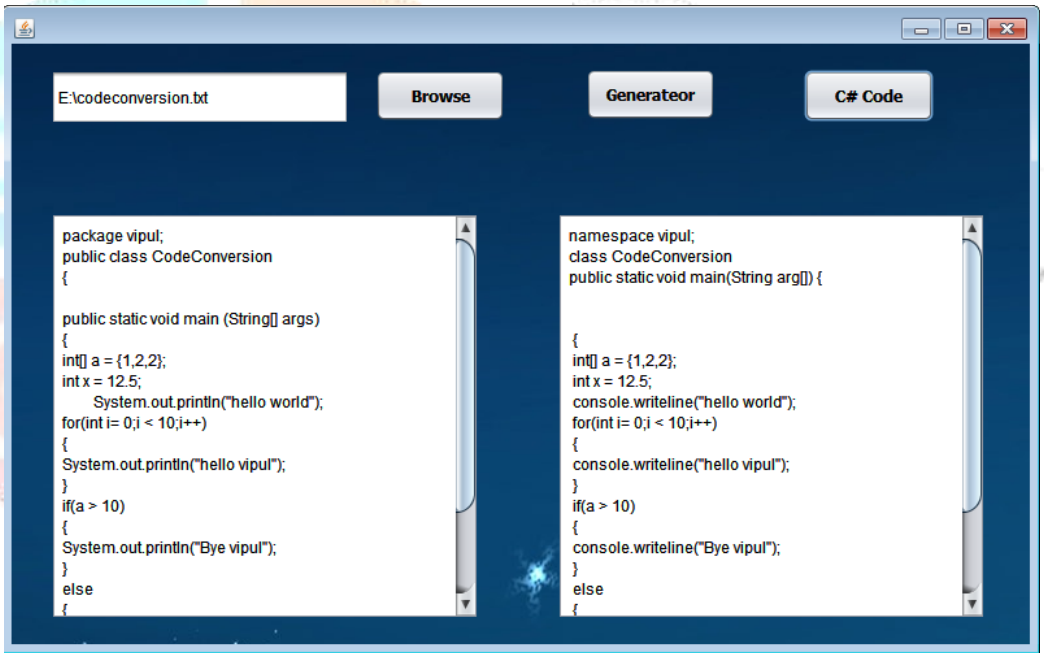
As the figure 3 above describes, the proposed system will get java code as input. The source code which has the keyword, class, variable etc. Then the initial part of the system will be getting executed. In the second phase main work is generating an XML file with defined keywords and attributes. Figure 4 dictates class functions have changed to new source code. After generating xml files apply some rules and patterns to file and generate final code. As figure 5 and 6 describes that the proposed system will also convert MySQL connectivity code of java to the equivalent code of C#. The statements such as Class.forName, DriverManager and other statements will be getting converted into the equivalent C# code. MySQL connectivity code contains various different statements which are stored in the database.



```
package vipul;
public class CodeConversion
{

public static void main (String[] args)
{
int[] a = {1,2,2};
int x = 12.5;
      System.out.println("hello world");
for(int i= 0;i < 10;i++)
{
System.out.println("hello vipul");
}
if(a > 10)
{
System.out.println("Bye vipul");
}
else
{
```

Figure 3: Code Input



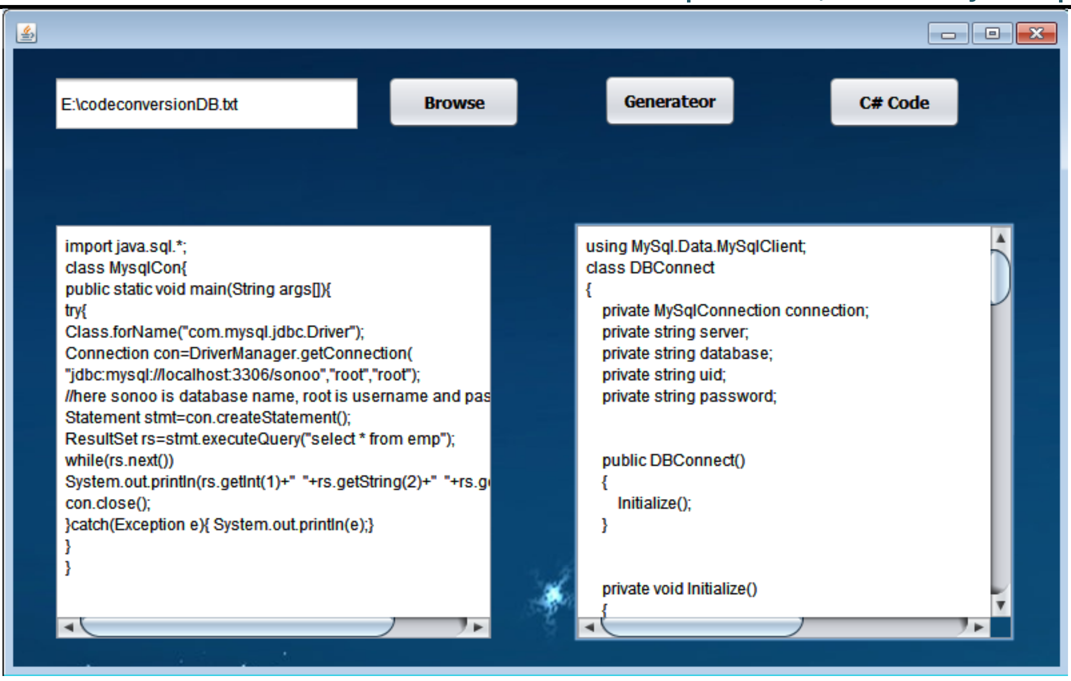
```
package vipul;
public class CodeConversion
{

public static void main (String[] args)
{
int[] a = {1,2,2};
int x = 12.5;
      System.out.println("hello world");
for(int i= 0;i < 10;i++)
{
System.out.println("hello vipul");
}
if(a > 10)
{
System.out.println("Bye vipul");
}
else
{
```

```
namespace vipul;
class CodeConversion
public static void main(String arg[]) {


{
int[] a = {1,2,2};
int x = 12.5;
console.writeline("hello world");
for(int i= 0;i < 10;i++)
{
console.writeline("hello vipul");
}
if(a > 10)
{
console.writeline("Bye vipul");
}
else
{
```

Figure 4: Mapping of code



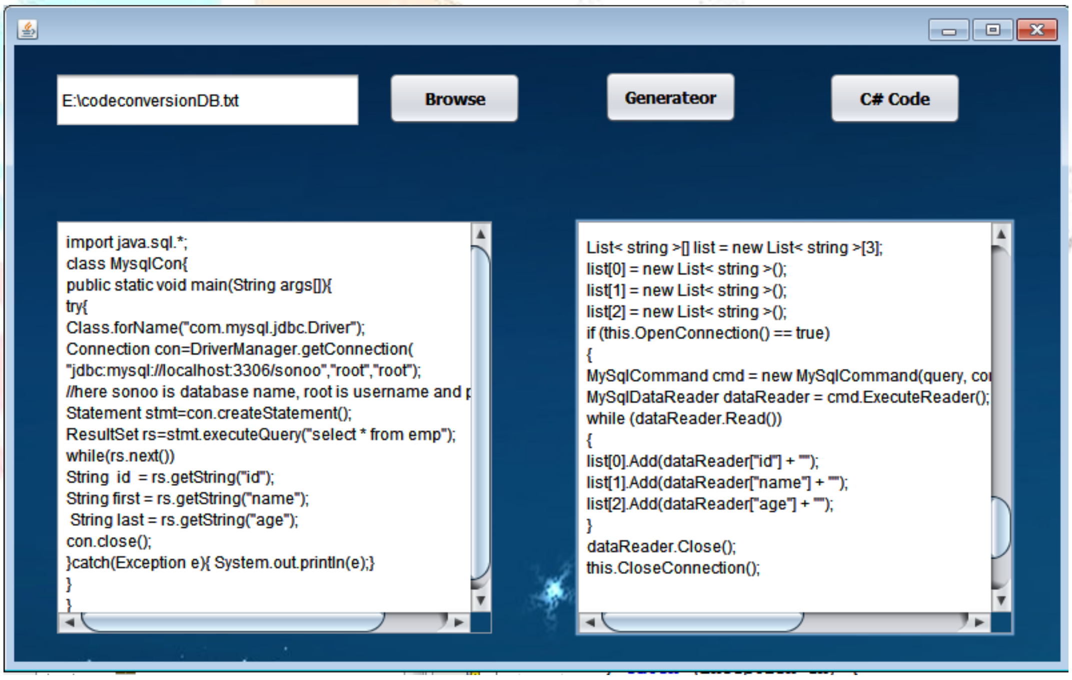
```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and pas
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.g
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

```
using MySql.Data.MySqlClient;
class DBConnect
{
   private MySqlConnection connection;
   private string server;
   private string database;
   private string uid;
   private string password;


   public DBConnect()
   {
      Initialize();
   }


   private void Initialize()
   {
```

Figure 5: Output Result in 1 C#



```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and p
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
String  id  = rs.getString("id");
String first = rs.getString("name");
 String last = rs.getString("age");
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

```
List< string >[] list = new List< string >[3];
list[0] = new List< string >();
list[1] = new List< string >();
list[2] = new List< string >();
if (this.OpenConnection() == true)
{
MySqlCommand cmd = new MySqlCommand(query, co
MySqlDataReader dataReader = cmd.ExecuteReader();
while (dataReader.Read())
{
list[0].Add(dataReader["id"] + "");
list[1].Add(dataReader["name"] + "");
list[2].Add(dataReader["age"] + "");
}
dataReader.Close();
this.CloseConnection();
```

Figure 6: Output Result 2 in C#

X. CONCLUSION AND FUTURE SCOPE.

The primary reasons for the language conversion is to transfer an application to a modern language or platform. The migration decision is getting affected by many factors that includes language and platform support, developers' availability, cost, performance and speed, market expansion and third party product availability. However, the supreme factor for making the decision is the availability of good tool support for the conversion; otherwise, only easy programs can be manually converted cost effectively. In this paper mentioned some different approaches to convert source code from one language to another (java to C#). The system has been developed greatly to reduce load and work in good fashion. It is palpable to the user that the effect of code conversion are good. In future it will be also attainable to convert to the complex code with different files.

XI. REFERENCES

[1] Microsoft, Java Language Conversion Assistant 3.0 (Beta), 2004.

[2] A. Terekov and C. Verhoef, The realities of language conversion, IEEE Software 2000, 17(6): 111 124, 2000.

[3] N. Nanda and S. Kumar, Migrating Java applications to .Net, JavaWorld Jan. 2003. Available at www.javaworld. com/javaworld/jw-01-2003/jw-0103-migration_p.html

[4] El-Kassas, Wafaa & Abdullah, Basem & Hassan Yousef, Ahmed & Wahba, Ayman. (2015). Taxonomy of Cross-Platform Mobile Applications Development Approaches. Ain Shams Engineering Journal. 8. 10.1016/j.asej.2015.08.004.

[5] H. Zhong, S. Thummalapenta, T. Xie, L. Zhang and Q. Wang, "Mining API mapping for language migration," *2010 ACM/IEEE 32nd International Conference on Software Engineering*, Cape Town, 2010, pp. 195-204.

[6] Klima P., Selinger S. (2013) Towards Platform Independence of Mobile Applications. In: Moreno-Díaz R., Pichler F., Quesada-Arencibia A. (eds) Computer Aided Systems Theory - EUROCAST 2013. EUROCAST 2013. Lecture Notes in Computer Science, vol 8112. Springer, Berlin, Heidelberg