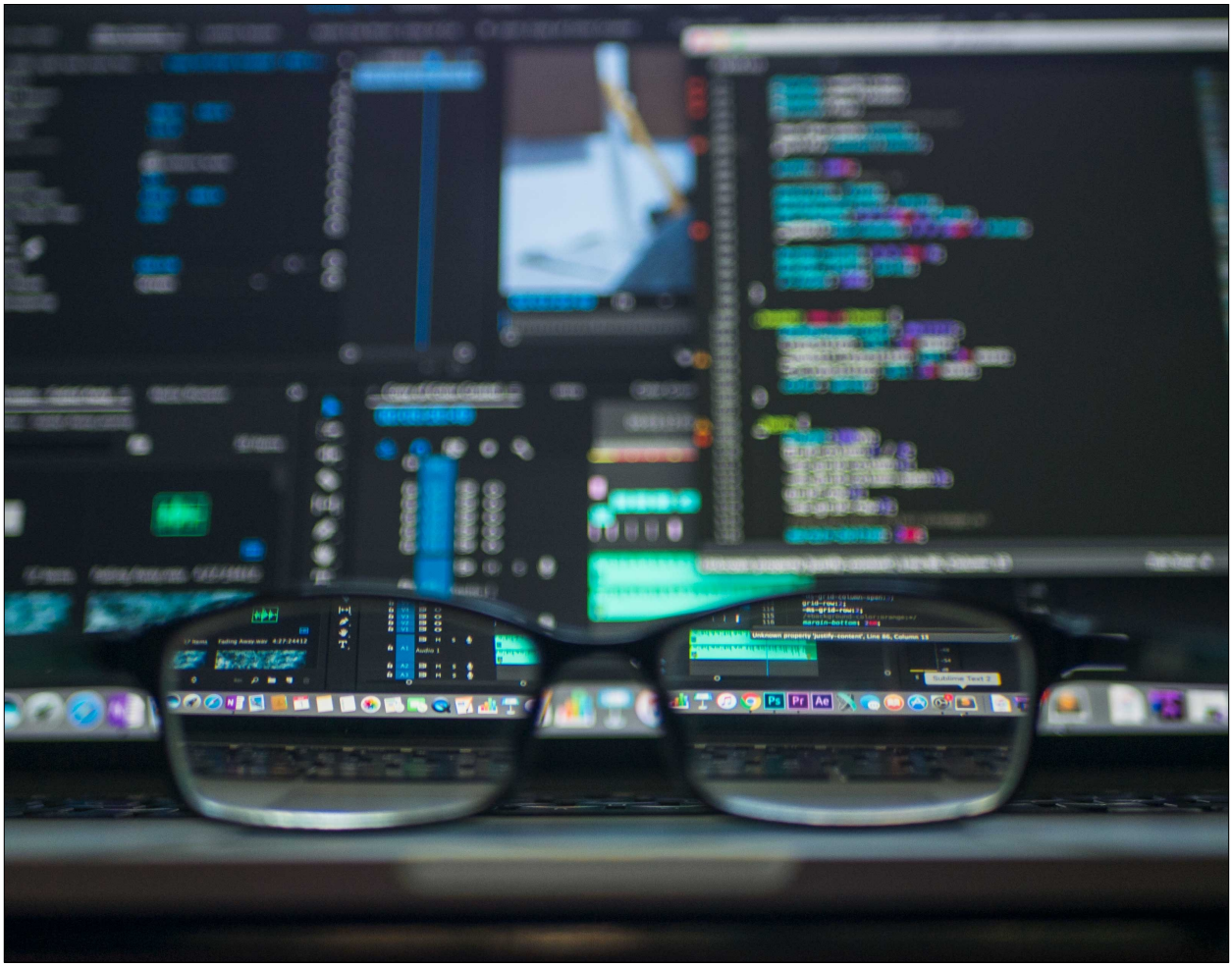
FEBRUARY 2021



ZeptoN Programming Language

White Paper

ZeptoN: Putting Program into Java

1. Introduction 5

2. ZeptoN Features 8

3. The Program Entity 11

4. ZeptoN Runtime 16

5. Example Programs 20

6. Future of ZeptoN 39

7. Grammar 40

8. References 41

9. Predefined Environment 42

Get the ZeptoN Compiler 45

About the Author 46

Copyright 47

License 47

Credits 50

1. Introduction

The ZeptoN programming language “Zep” is a programming language that is based on Java syntax, and compiles to a Java bytecode .class file. But unlike Java, ZeptoN is easy to use and to learn, which is the primary purpose of the programming language.

ZeptoN is an imperative-type programming language that generates bytecode to run on the JVM (Java Virtual Machine) and thus it can be used anywhere Java is used today.

1.1 Motivation

The motive for developing and creating the ZeptoN programming language is to create a language that a user can focus on packages and libraries to ***do*** something, and not always creating a class, prototype, interface, unless you really need or want to in writing the code.

Graham states as one of his points [Grah 2001] is that, "Design for Yourself and Your Friends." This is what I have done with ZeptoN, but rather than starting from scratch, I leveraged the existing language Java to have existing syntax to use. Graham further explains [Grah 2001]:

> *"If you look at the history of programming languages, a lot of the best ones were languages designed for their own authors to use, and a lot of the worst ones were designed for other people to use."*

Once in college, I e-mailed the creator of C++, Bjarne Stroustrup, with a question about the maxim "Everything is an object," in the object-oriented paradigm and such programming languages. Stroustrup responded that everything can be a class, or is a class. I disagree, but in programming languages especially Java, everything is a class– which can be quite tedious.

Thus I wanted a language with a program, but also one that I can leverage and use existing infrastructure to avoid re-creating the wheel for a completely new programming language. Thus I created the ZeptoN programming language. Voila!… or Quod Erat Demonstrandum.

1.2 Users

There are two types of users of the ZeptoN programming language envisioned by its creator:

1. Newb - “newbie” learning to program and write code
2. Coder - an experienced practitioner in programming

These two types of users seem extreme opposite ends of the continuum, but paradoxically, ZeptoN is idyllic for both kinds of users, and those in the middle of the continuum.

1.2.1 Newb

A "Newb" or “Noob” is a newbie, a person learning to program, or learning to program in a C-style syntax programming language. A developer experienced in Pascal, Ada, or FORTRAN is an example is a newb to ZeptoN, and can focus on learning syntax and statements without the complexity of the object, class, and other object-oriented concepts and abstractions. A student in school, university, or in company training would be a “Newb” learning to program.

1.2.2 Coder

The coder is experienced and with C-syntax style languages such as C, C++, or C# but wants to learn the Java packages and libraries. A coder also wants to be able to write a quick-and-dirty programs using Java packages and libraries without the tedium of creating a class, constructors, etc. A data engineer might want to easily write a quick, simple ZeptoN program using some Java statistical packages.

1.2.3 Get It Done

ZeptoN is a programming language that is meant to get things done. While a complex object-oriented design pattern is nice in a library, sometimes the end source code that is written, is simply trying to achieve a result or object, quickly and efficiently.

Paul Graham [Grah 2001] makes a point "A programming language has to be good for writing throwaway programs" stating:

> *You know what a throwaway program is: something you write quickly for some limited task. I think if you looked around you'd find that a lot of big, serious programs started as throwaway programs. I would not be surprised if most programs started as throwaway programs. And so if you want to make a language that's good for writing software in general, it has to be good for writing throwaway programs, because that is the larval stage of most software.*

Thus ZeptoN is for the newb wanting to learn by doing, and an experienced developer writing a program to test a new idea, prototype a feature, or a technically savvy person writing a program to help in their non-developer work.

2. ZeptoN Features

The ZeptoN programming language has the following features:

1. Simple
2. Minimal
3. Coherent
4. Compatible
5. Familiar
6. Sans Object
7. Robust
8. Secure
9. Architecture Aloof
10. Portable
11. Open

Simple

ZeptoN is simple, as there is only one necessary entity to learn to use the language– the program structural entity. Writing a ZeptoN program uses the program entity structure for all functionality.

Minimal

ZeptoN is a minimalist programming language by design. ZeptoN uses and builds upon Java syntax, but only adds two keywords–**prog** for program, and **begin** for the central method of execution.

Coherent

ZeptoN is a coherent programming language--logical and consistent, clearly and carefully designed. ZeptoN has the program entity, and also the predefined

environment but with the single criterion of consistent with Java, and inclusion of Java features.

Compatible

ZeptoN is compatible with Java syntax completely, and with uses many commonly used functions in packages as part of the predefined environment available to the user. ZeptoN is also binary compatible the generated bytecode .class file will run on a Java Virtual Machine on any platform.

Familiar

ZeptoN is familiar in the syntax used is the exact same as Java, and familiar to programmers who have used C, C++, and C#. This familiarity allows easy learning for an experienced developer, and easily transition to other programming languages, obviously Java, for the newbie.

Object Optional

ZeptoN is optional object, meaning that having to use object-oriented principles or functionality such as a class, a constructor, access to a class, and so forth is optional to write ZeptoN programs—if you want to use though, you can.

Robust

ZeptoN transcompiles into Java source code, which is then compiled into a bytecode .class file. ZeptoN is strongly typed, and checked by the underlying Java compiler. Also source code is woven into a ZeptoN program to catch and gracefully handle errors and exceptions at runtime.

Secure

ZeptoN utilizes the bytecode verification and runtime management system to be secure. This is done opaquely so the programmer does not have to worry about the infamous bugs, glitches, and defects that have led to security holes in software.

Architecture Aloof

ZeptoN, building upon Java is architecture aloof. Software does not depend exclusively on the features of the underlying hardware, so there is no “lock into” the hardware, or operating system. A ZeptoN program can inter-operate with other platforms, systems, servers completely unaware of the architecture details.

Portable

ZeptoN is portable as it uses the Java Virtual Machine and compiles into a bytecode .class file. Portability is also possible because ZeptoN is architecture neutral, so can work with other systems easily without being “ported” and updated to run, work, or inter-operate with another system.

Open

The ZeptoN programming language, the transcompiler is implemented in Java and ZeptoN, and is completely open. The source code is open source, the license is the GNU Public License (GPL) version 3.0. Thus ambitious and smart developers can build and tinker with the transcompiler.

3. The Program Entity

The only, and primary entity in ZeptoN is not a class or object, but a program. Other entities can be declared and used, but the program is the primary entity—a unit of execution, not re-use.

3.1 Program Entity

The program entity has only two aspects:

1. Program body to declare methods and attributes
2. Program block to declare statements in central method of execution

3.1 Program Syntax Elements

There are three syntax elements of a program:

1. Program header declares the program with an identifier
2. Begin divider declares the executable source code
3. Program footer finishes the program declaration

The program entity syntax is:

```
prog <IDENT> {       //program header

//PROGRAM_BODY

  begin {

  //PROGRAM_BLOCK

  }
}                    //program footer
```

The program header begins the program, and the program footer closes the program declaration and definition. The keyword "**begin**" separates or divides the program body from the program block. The program body is optional, but the program block is required in a ZeptoN program.

3.2 Program Body

The program body is two-fold around the program header. Before the program header, a namespace can be declared, along with any packages or libraries included. ZeptoN automatically includes some packages and libraries, so that they simply can be used.

After the header is the program body where attributes and methods of the program are declared. Attributes and methods that are elements of the program are essentially of two kinds:

1. Static - or singular, these program elements are singleton, stateless accessible without a dynamic instance.

2. Non-static - or instance, these program elements are bound to a dynamic instance that is created through which they are accessed.

3.2.1 Program Methods

Methods of a program are declared like C-style functions, and are either static (stateless) or non-static (instance) methods. The method declares a return type, or void is followed by the method name, any parameters. Then within opening and closing braces are the program statements.

```
void instanceFunc(parameters){
  //...statements
}//end instanceFunc
```

The static method is declared similarly to an instance method, a C-style function but with the keyword ‘static’ to indicate a static method.

```
static void staticFunc(parameters){
  //...statements
}//end staticFunc
```

3.2.2. Program Attributes

Attributes of the program are constants which are immutable, and variables which are mutable. These attributes are visible and accessible within the entire program—by program methods, and the program block.

3.2.2.1 Constants

A program constant is like a program method, in that there is a static and non-static (instance) attribute. A constant is declared with the keyword "final" and then if static, the keyword "static" followed by the type, the naming identifier, and the value for the constant.

```
final static double PI = 3.14159265358979323846;//constant immutable static

final int ZERO = 0; //constant immutable instance
```

3.2.2.2 Variables

A program variable is like a program method, in that there is a static and non-static (instance) variable. A static variable is declared with the keyword "static" before the variable definition. A type followed by the naming identifier, and then an initial value for the variable.

```
static long totalValue = 0L;

boolean isFileReady = false;
```

3.3 Program Block

The program block is declared and divided with the keyword "begin" which also closes the program body.

The program block is the central nexus of execution, the program is executed starting in the program block. The program block is like a static method, only declared without a naming identifier, or any explicit parameters passed.

Within the program block, are the statements that are executed as the first method in the program.

```
  begin {
    //statements
  }
}//end prog
```

A program can possibly have no program body and simply be a program block.

```
prog <IDENT> {
  begin {
    //PROGRAM_BLOCK
  }//end begin
}//end prog <IDENT>
```

Hence, it is very easy to implement and execute a quick-and-dirty ZeptoN program.

3.4 Classes

ZeptoN does not use classes, interfaces, or objects as a first-class entity in writing source code. However, ZeptoN supports inner classes that can be declared and used within a ZeptoN program for a more sophisticated user and much more complex program. But ZeptoN is usable without any requirement for a class, interface, or object.

4. ZeptoN Runtime

The ZeptoN runtime is a predefined environment as a set of imports, constants and methods automatically provided as static, immutable methods. There are 76-methods, 4-constants, 1-special constant variable, and 6-imports provided in the runtime environment.

4.1 Imports

ZeptoN also automatically imports several Java packages by default. This allows the user to use other standard packages from Java not provided explicitly as a method or constant. The auto-magic imports are:

1. java.io – input/output
2. java.lang - the Java language infrastructure
3. java.math – mathematics
4. java.nio.charset - character sets
5. java.net – networking
6. java.util – utility and collections

4.1 Constants

There are four constants, three static, and one non-static (instance) immutable variable.

4.1.1 Static Constants

ZeptoN provides three predefined static constants for readability:

1. EMPTY_CHAR – an empty char '' character.
2. EMPTY_STRING – an empty string "" of characters.
3. EOL – end of line or line separator string for the platform
4. NULL_CHAR – the '\0' null character.

4.1.2 Magical me

ZeptoN provides one predefined immutable or constant variable (an automatically created instance reference to the program itself) as an immutable program block constant using the identifier **me**.

The constant variable **me** represents a reference to the current instance of the program within the program block. This allows access to all methods and attributes of the program, but ***only*** in the program block. As the variable **me** is immutable, it cannot be assigned to in the program block, nor accessed outside the program block. An example illustrating the **me** constant instance variable is:

```
 prog badMagicalMe {
 begin {

   nop();

   me = EMPTY_STRING;

   exit(0);
 }//end begin
}//end prog badMagicalMe
```

When compiled the compiler output is:

```
 Error: badMagicalMe.java.
 Line 16 At 3: cannot assign a value to final variable me
   me = EMPTY_STRING;
   ^

 Error: badMagicalMe.java.
 Line 16 At 8: incompatible types
   me = EMPTY_STRING;
        ^
```

4.2 Methods

The methods are widely used in Java from different packages and libraries, only there is no need to import, declare, and instantiate the packages and libraries. The methods for input, output, conversion to String are the most common predefined methods:

```
static void print(...)
static void println(...)
static void printf(...)
static String toString(...)

static String readLine()
static char[] readPassword()
static int readInt()
```

Others are platform and utility functions:

```
static void gc()
static void exit(...)
static void arraycopy(...)
static long nanoTime()
static long freeMemory()
static Console getConsole()
```

All of the methods are exactly the same for parameters and functionality to the methods in the Java packages and libraries. Three ZeptoN specific methods in the predefined environment are:

```
static String[] getArgs();
void errorf(...);
void nop();
```

The method “getArgs()” gets the command-line arguments. The method “errorf()” method prints to the error stream. The method “nop()” is a no operation method for an empty or nothing statement, instead of a simple, less explicit semicolon.

5. Example Programs

The syntax of exemplar programs illustrates actual source code of ZeptoN.

5.1 Ubiquitous “Hello, World!!!”

The first six examples, are the classic, and ubiquitous “Hello, World!!!” program. The same program that prints “Hello, World!!!” Is implemented in six different ways to illustrate the syntax and features of a ZeptoN program.

5.1.1 Simple Program

The simplest ZeptoN program has a program block is:

```
prog helloWorld {
  begin {
    println("Hello, World!!!");
    exit(0);
  }
}//end prog helloWorld
```

5.1.2 Program with Static

A more complex ZeptoN program has a static constant attribute, and a static method to print the text of "Hello, World!!!" output. The ZeptoN program with static elements is:

```
prog helloWorld {

  final static String HELLO_WORLD = "Hello, World!!!";

  static void HelloWorld(){
    println(HELLO_WORLD);
  }//end HelloWorld

  begin {
    HelloWorld();
    exit(0);
  }//end begin
}//end prog helloWorld
```

5.1.3 Program with an Instance

A ZeptoN program with non-static (instance) elements of a constant attribute, and a method is:

```
prog helloWorld {
  final String HELLO_WORLD = "Hello, World!!!";

  void HelloWorld(){
    println(this.HELLO_WORLD);
  }//end HelloWorld

  begin {
    me.HelloWorld(); //use predefined immutable instance variable 'me'
    exit(0);
  }//end begin
}//end prog helloWorld
```

The ZeptoN program uses the predefined immutable variable “me” to access the non-static method, which then accesses the static constant attribute.

5.1.4 Program with Class

A ZeptoN program using an internal, inner class uses the predefined immutable variable “me” to instantiate the class, to invoke the method.

```
prog helloWorldWithClass {

  class HelloWorldClass {
    void helloWorld(){
      println("Hello, World!!!”);
    }//end helloWorld
  }//end class helloWorldClass

  begin {
    helloWorldWithClass.helloWorldClass say = me.new helloWorldClass();
    say.HelloWorld();
    exit(0);
  }//end begin
}//end prog helloWorldWithClass
```

5.1.5 Program with an Enumeration

A ZeptoN program using an internal, enumeration uses the values of the enumeration.

```
prog helloWorldWithEnum {

  enum GreetingHelloWorld {
    HELLO("Hello"), COMMA(","), WORLD("World"), BANG3("!!!");

    final String text;

    GreetingHelloWorld(String text) {
      this.text = text;
    }//end enum constructor
  }//end enum

begin {

    for(GreetingHelloWorld sayGreeting : GreetingHelloWorld.values()){
      print(sayGreeting.text);
      if(sayGreeting != GreetingHelloWorld.HELLO)
        print(" ");
    }//end for

    println();
    exit(0);
  }//end begin
}//end prog helloWorldWithEnum
```

5.1.6 Program using Static Array

A ZeptoN program with non-static (instance) element of a constant attribute of an array is:

```
prog helloWorldUseStaticArray {

  final static String[] GREETING = { "Hello", ",", "World", "!!!" };

begin {

    for(String greet : GREETING){
      print(greet);
      if(!greet.equals("Hello"))
        print(" ");
    }//end for

    println();

    exit(0);

  }//end begin

}//end prog helloWorldUseArray
```

5.2 Medley of Examples

These example ZeptoN programs are a medley of different types illustrating the source code, syntax, and structure of a ZeptoN program.

5.2.1 Mystery Program

The program "Mystery.zep" compiles and runs, but is a do nothing program. The question is why, and that is the mystery.

```
prog Mystery {

  begin {

    nop();
    https://wgilreath.github.io/WillHome.html
    exit(0);

  }//end begin
}//end prog Mystery
```

The program "Mystery.zep" compiles and runs, as the obvious URL is a label and a comment, therefore nothing in terms of statements of source code.

5.2.2 Convert Fahrenheit to Celsius

The simple program "ConvertFarenToCelsius.zep" converts temperatures from Fahrenheit to Celsius from freezing to boiling.

```
prog ConvertFarenToCelsius {
  final static double RATIO = 5.0d/9.0d;
begin {
    int tempCelsius = 0;
    println("Fahrenheit           Celsius");
    println("---------------------------------------");
    for(int tempFahren = 32; tempFahren <= 212; tempFahren++){
      tempCelsius = (int) (RATIO * ((double) (tempFahren - 32)));
      printf("%3d-degrees Fahren %3d-degrees Celsius%n", tempFahren, tempCelsius);
    }//end for
    println();
    exit(0);
  }//end begin
}//end prog ConvertFarenToCelsius
```

5.2.3 Dummy Do-While Loop

The "dummy" do-while loop program "DummyDoWhileLoop.zep" creates a labeled block using a do-while loop that executes only once. This allows for potential branching to and from the label using a break or continue statement.

```
prog DummyDoWhileLoop {
  begin {

    label: do { //execute do-while loop once
      nop();
      println("Inside dummy do-while loop.");
    } while(false);

    exit(0);
  }//end begin
}//end prog DummyDoWhileLoop
```

5.2.4 Fetch Uniform Resource Locator (URL) Online

The program “FetchURL.zep” will fetch or retrieve the contents at a Uniform Resource Locator (URL) and dump the contents to the console or terminal window.

```
package will.zepton.demo;

prog FetchURL {
begin {

  final StringBuilder content = new StringBuilder(EMPTY_STRING);

  try {

    URL url = new URL("https://wgilreath.github.io/WillHome.html");
    URLConnection urlCon = url.openConnection();
    println(url.toString());
    BufferedReader reader = new BufferedReader(new
                            InputStreamReader(urlCon.getInputStream()));
    String line = EMPTY_STRING;
    while((line = reader.readLine()) != null){
      content.append(line + "\n");
    }//end while

    println(content);
    buffered.close();
  }catch(Exception e) {
    e.printStackTrace();
  }//end try

  exit(0);
  }//end begin
}//end prog FetchURL
```

5.2.5 Towers of Hanoi

Another example ZeptoN program is “TowersOfHanoi.zep” that illustrates recursion, user input, and output in the classic programming problem. The implicit immutable variable “me” is used to access the non-static instance attribute and method.

```
prog TowersOfHanoi {

  int disks = -1;

  void hanoi(char src, char tmp, char dst, int n){
    if(n > 0){
      this.hanoi(src, tmp, dst, n-1);
      printf("  Move disk %d from peg %c to peg %c. %n", n, src, dst);
      this.hanoi(tmp, src, dst, n-1);
    }//end if

  }//end hanoi

begin {

    print("Enter the number of disks: ");
    me.disks = readInt();
    println();
    println("Towers of Hanoi Solution: ");
    me.hanoi('A','B','C', me.disks);

    exit(0);

  }//end begin
}//end prog TowersOfHanoi
```

5.2.6 N-Queens Problem

The N-queens problem is a classic problem in recursion and backtracking to solve the problem of placing chess queens on a NxN chess board so that no queen can attack the other. The chess board must be at least N = 4 for any possible solution to exist.

```
prog NQueensProblem {

  int N  = -1;

  void printSolution(int board[][]){
    printf("%nN = %d%n", board.length);
    print("--");
    for(int x=0;x<N;x++) print("---");
    println();
    for(int i = 0; i < N; i++) {
      print("|");
      for(int j = 0; j < N; j++){
        if(board[i][j] == 0){
          printf(" _ ");
        } else
          printf(" Q ");
      }//end for
      print("|");
      println();
    }//end for
    for(int x=0;x<N;x++) print("---");
    print("--");
    println();
  }//end printSolution

  boolean isSafe(int board[][], int row, int col){
    int i, j;
    for(i = 0; i < col; i++)
      if(board[row][i] == 1) return false;
    for(i = row, j = col; i >= 0 && j >= 0; i--, j--)
      if(board[i][j] == 1) return false;
    for(i = row, j = col; j >= 0 && i < N; i++, j--)
      if(board[i][j] == 1) return false;
    return true;
  }//end isSafe
```

```
  boolean solveNQ(int board[][], int col){
    if(col >= N) return true;
    for(int i = 0; i < N; i++) {
      if(isSafe(board, i, col)) {
        board[i][col] = 1;
        if(solveNQ(board, col + 1)) return true;
        board[i][col] = 0;
      }//end if
    }//end for
    return false;
  }//end solveNQ

  boolean solveNQ(){
    int board[][] = new int[N][N];
    for(int x=0;x<N;x++){
      for(int y=0;y<N;y++){
        board[x][y] = 0;
      }//end for
    }//end for
    if(!solveNQ(board, 0)){
      print("Solution does not exist");
      return false;
    }//end if
    printSolution(board);
    return true;
  }//end solveNQ

begin {

    for(int x=4;x<=10;x++) {
      me.N = x;
      me.solveNQ();
    }//end for
    exit(0);

  }//end begin
}//end prog NQueensProblem
```

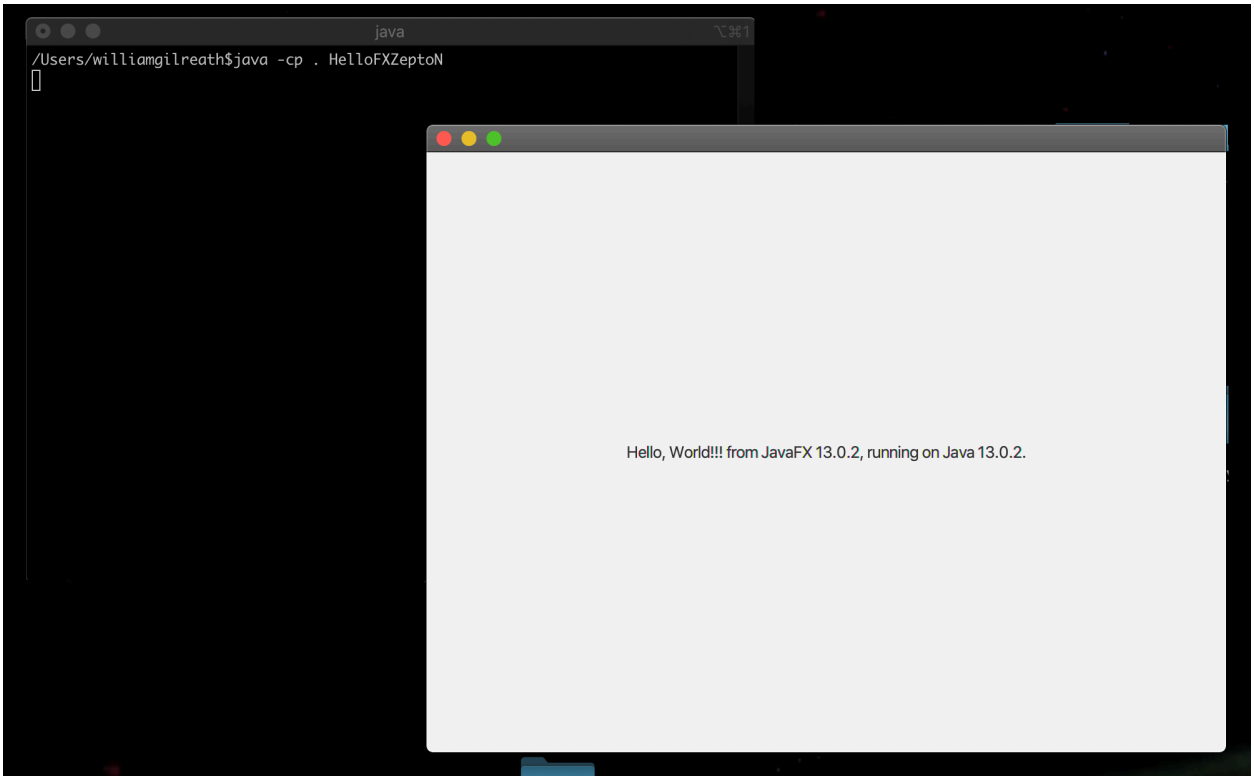
The ZeptoN program “NQueensProblem.zep” iterates from 4 to 10 for the size of N, and computes the arrangement solution, which is then printed to the terminal or console.

5.2.7 Gooey Hello

Another "Hello, World!!!" ZeptoN program uses the Java graphic user interface library, JavaFX to create a window with the greeting, and the Java runtime version, and JavaFX version.

```
import javafx.application.Application;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.scene.Scene;
import javafx.stage.Stage;

prog HelloFXZeptoN {

  static class Window extends Application {

    @Override
    public void start(Stage stage) {

      String javaVer   = getProperty("java.version");
      String javaFXVer = getProperty("javafx.version");

      Label label = new Label("Hello, World!!! from JavaFX " + javaFXVer
                              + ", running on Java " + javaVer + ".");
      Scene scene = new Scene(new StackPane(label), 640, 480);
      stage.setScene(scene);
      stage.show();

    }//end start

  }//end class Window

  begin {

    Application.launch(Window.class, getArgs());

  }//end begin
}//end prog HelloFXZeptoN
```

When run, the ZeptoN program "HelloFXZeptoN.zep" creates the following window on the desktop as the screenshot illustrates:

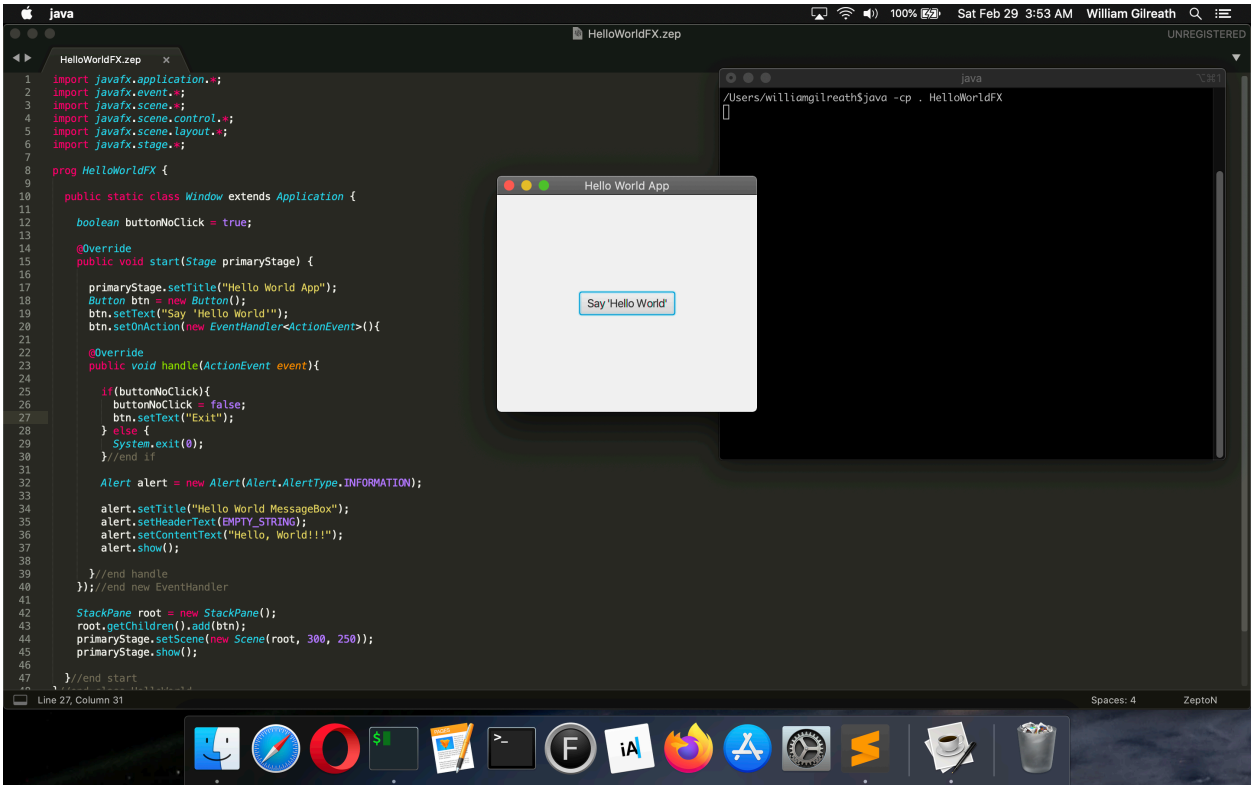


HelloFXZepton.zep ZeptoN Program Running…

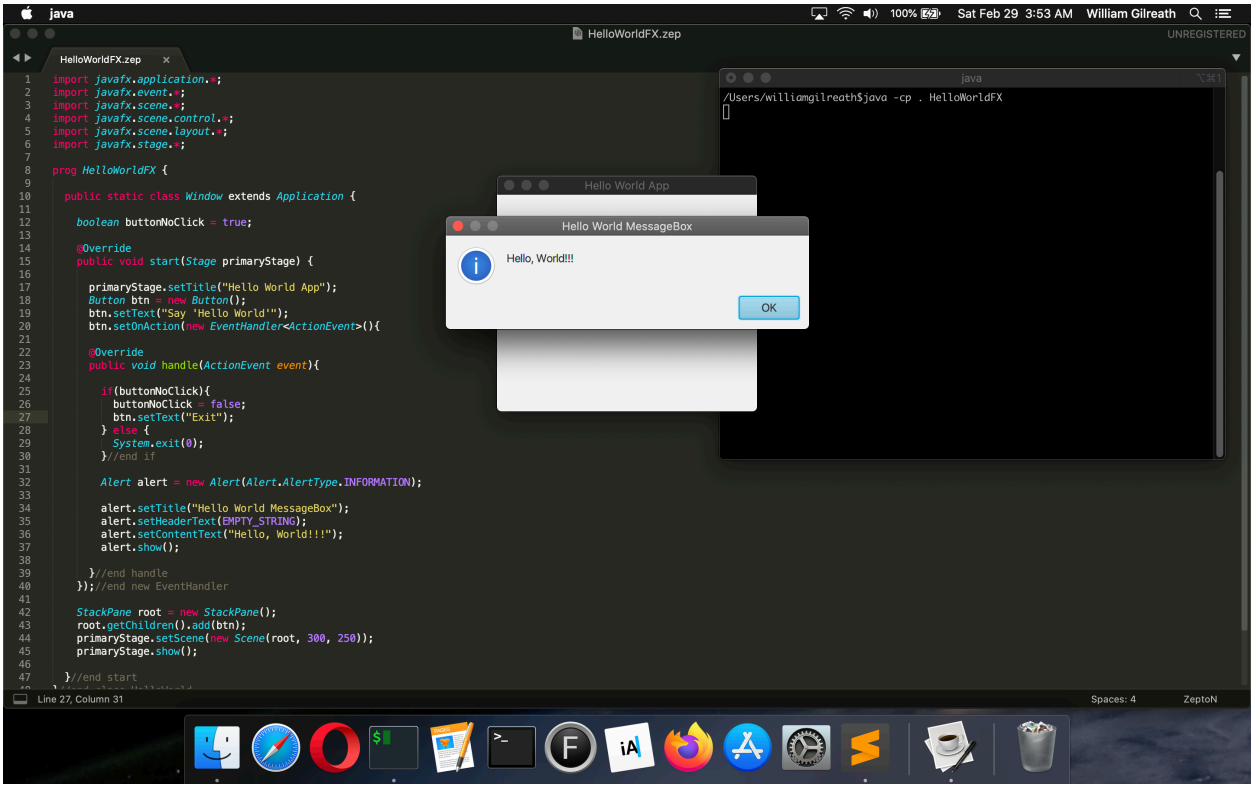
The "HelloFXZepton.zep" demonstrates that ZeptoN can utilize the existing Java libraries and packages. Creating an inner class with explicit accessible methods is used, but ZeptoN, while "sans object" does not stop the user from using a class, object-oriented features if they wish.

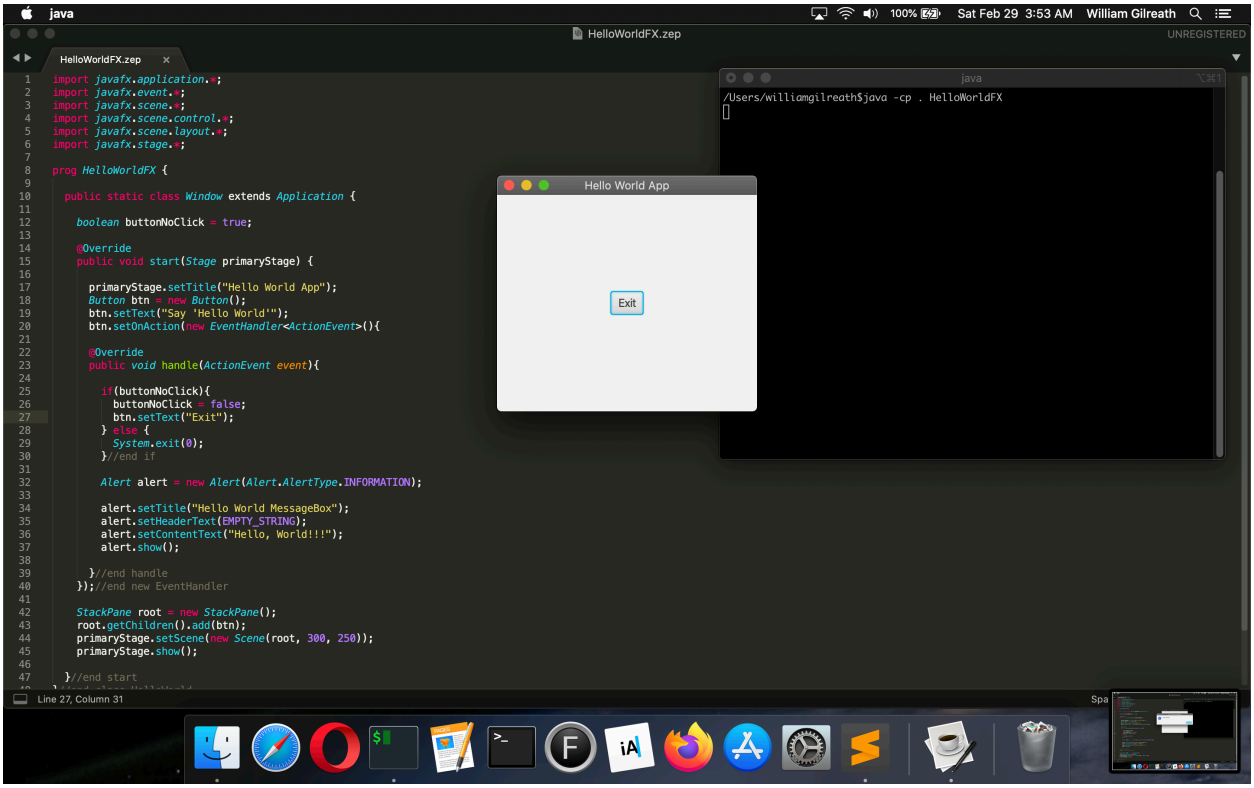
5.2.8 Gooey Message Box Greeting

A more sophisticated example is the ZeptoN program “HelloWorldFX.zep” which creates a pane with a button “Say ‘Hello World!!!.’

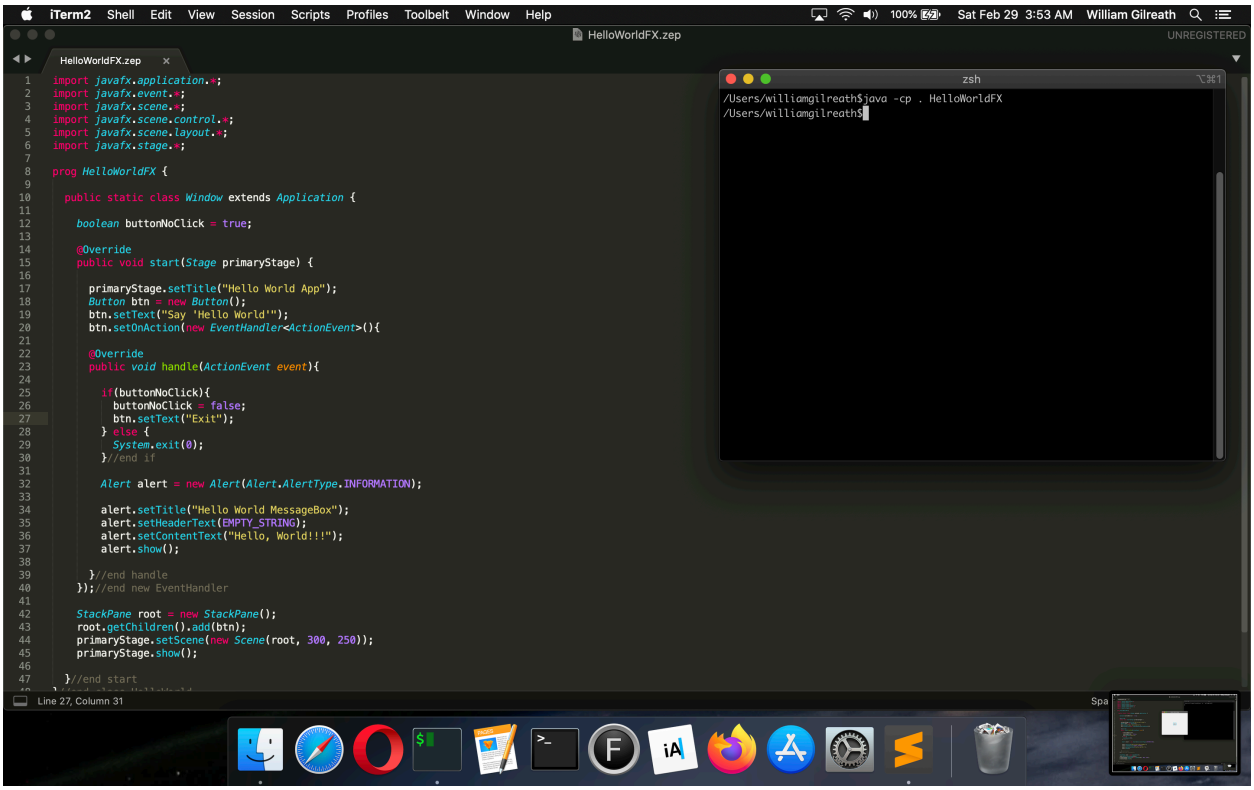


When the button is clicked it shows a message box with “Hello World!!!” but also changes the button to “Exit.”





When the button is clicked, the ZeptoN program exits.



The screenshots show how a more complex event handling visual program using JavaFX is possible in ZeptoN.

The source code for “HelloWorldFX.zep” is:

```
import javafx.application.*;
import javafx.event.*;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.*;

prog HelloWorldFX {

  public static class Window extends Application {

    boolean buttonNoClick = true;

    @Override
    public void start(Stage primaryStage) {

      primaryStage.setTitle("Hello World App");
      Button btn = new Button();
      btn.setText("Say 'Hello World'");
      btn.setOnAction(new EventHandler<ActionEvent>(){

      @Override
      public void handle(ActionEvent event){

        if(buttonNoClick){
          buttonNoClick = false;
          btn.setText("Exit");
        } else {
          exit(0);
        }//end if

        Alert alert = new Alert(Alert.AlertType.INFORMATION);

        alert.setTitle("Hello World MessageBox");
        alert.setHeaderText(EMPTY_STRING);
        alert.setContentText("Hello, World!!!");
        alert.show();

      }//end handle

    });//end new EventHandler
```

```
    StackPane root = new StackPane();
    root.getChildren().add(btn);
    primaryStage.setScene(new Scene(root, 300, 250));
    primaryStage.show();

  }//end start

}//end class HelloWorld

begin {

    Application.launch(Window.class, getArgs());

  }//end begin
}//end prog HelloWorldFX
```

The ZeptoN program "HelloWorldFX.zep" uses an internal class, and the JavaFX package and library to create a simple graphic user interface program. Using attributes and methods of the predefined environment simplifies the source code.

6. Future of ZeptoN

The future of the ZeptoN programming language is two-fold in the areas of:

1. Java
2. ZeptoN

The future of ZeptoN as ZeptoN builds on Java, follows Java in as new features and concepts are added, they are implicitly and automatically added to ZeptoN.

As of this writing (April 2020) Java in the next JDK 14, will add a record entity, and a pre-release feature is block strings that will become part of Java as well. Both will be supported by ZeptoN once a part of the programming language.

ZeptoN will also in and of itself, grow, and evolve. For example, new static attributes and methods in the predefined environment can be added, and removed depending upon the community of ZeptoN users. Possibly standard, simpler packages and libraries can be added and used as part of the ZeptoN predefined environment.

Both ZeptoN and Java will grow in tandem, the language becoming more sophisticated with new features and capabilities over time.

7. Grammar

The ZeptoN programming language builds upon Java, with a program as the structural entity, and a program block as the central point of execution. Thus ZeptoN uses the Java grammar, but with the following modifications from the Java Language Specification 14 [Orac 2020]. The modifications to the grammar are for the following syntax rules:

```
CompilationUnit:
  ProgramCompilationUnit

ProgramCompilationUnit:
  [PackageDeclaration] {ImportDeclaration} ProgramDeclaration

ProgramDeclaration:
  ProgDeclaration

ProgDeclaration:
  prog TypeIdentifier ProgBody

ProgBody:
  {ClassBodyDeclaration} begin ProgBlock

ProgBlock:
  Block
  ;

PrimaryNoNewArray:
  me
  Literal
  ClassLiteral
  this
  TypeName . this
  ( Expression )
  ClassInstanceCreationExpression
  FieldAccess
  ArrayAccess
  MethodInvocation
  MethodReference
```

8. References

- [Grah 2001] Graham, Paul. "Five Questions about Language Design," https://www.paulgraham.com/langdes.html, May 2001, Accessed February 3, 2020.

- [Orac 2019] Oracle America, Inc., *The Java Language Specification*, 14th edition, https://docs.oracle.com/javase/specs/jls/se14/jls14.pdf. Accessed March 22, 2020.

9. Predefined Environment

A list of the seventy-six predefined functions in the standard ZeptoN environment is given in alphabetical order by method name. These methods are used in other packages and libraries, but for ZeptoN are consolidated as static methods available for use without any need to import, instantiate, etcetera.

```
1.   void       arraycopy(Object,int,Object,int,int)
2.   int        availableProcessors()
3.   String     clearProperty(String)
4.   Console    console()
5.   long       currentTimeMillis()
6.   CharSet    defaultCharSet()
7.   void       errorf(String,Object...) //ZeptoN specific
8.   void       exit(int)
9.   String[]   getArgs()  //ZeptoN specific
10.  String     getenv(String)
11.  Locale     getLocale()
12.  String     getProperty(String)
13.  Runtime    getRuntime()
14.  void       gc()
15.  long       freeMemory()
16.  void       halt(int)
17.  int        identityHashCode(Object)
18.  String     lineSeparator()
19.  long       maxMemory()
20.  long       nanoTime()
21.  void       nop()   //ZeptoN specific
22.  void       print(BigDecimal)
23.  void       print(BigInteger)
24.  void       print(boolean)
25.  void       print(byte)
26.  void       print(char)
27.  void       print(char[])
28.  void       print(double)
29.  void       print(float)
30.  void       print(int)
31.  void       print(long)
```

```
32.  void       print(Object)
33.  void       print(short)
34.  void       print(String)
35.  void       printf(String,Object...)
36.  void       println()
37.  void       println(BigDecimal)
38.  void       println(BigInteger)
39.  void       println(boolean)
40.  void       println(byte)
41.  void       println(char)
42.  void       println(char[])
43.  void       println(double)
44.  void       println(float)
45.  void       println(int)
46.  void       println(long)
47.  void       println(Object)
48.  void       println(short)
49.  void       println(String)
50.  BigDecimal readBigDecimal()
51.  BigInteger readBigInteger()
52.  boolean    readBoolean()
53.  byte       readByte()
54.  char       readChar()
55.  double     readDouble()
56.  float      readFloat()
57.  int        readInt()
58.  String     readLine(String,Object...)
59.  String     readLine()
60.  long       readLong()
61.  char[]     readPassword()
62.  char[]     readPassword(String,Object...)
63.  short      readShort()
64.  String     readString()
65.  String     setProperty(String,String)
66.  long       totalMemory()
67.  String     toString(boolean[])
68.  String     toString(byte[])
69.  String     toString(char[])
70.  String     toString(double[])
71.  String     toString(float[])
72.  String     toString(int[])
```

```
73.  String     toString(long[])
74.  String     toString(Object[])
75.  String     toString(short[])
76.  String     valueOf(char[])
```

This list of function methods includes the three ZeptoN-specific methods, and the other methods found in Java packages. Over time the predefined environment will change, with new methods and attributes added, and others deprecated.

Get the ZeptoN Compiler

The ZeptoN "project site is on GitHub at: ***https://wgilreath.github.io/ZeptoN/***

Downloads

The ZeptoN "Echo" transcompiler, source code editor Zeptor, and ZeptoN code examples, are available for download from the following links:

1. Zeptor the ZeptoN Code Editor

- https://bit.ly/2K9RfAM (v.1.3.3)

2. ZeptoN Code Editor in Action Video (with zither music)

- https://bit.ly/2xf4ddJ (Youtube! video of my code editor)

3. ZeptoN Code Examples

- https://bit.ly/3ebLSiF

4. ZeptoN "Echo" Compiler Binary

- https://github.com/wgilreath/ZeptoN/raw/master/ZepC.jdk8.jar

5. ZeptoN "Echo" Compiler Source Code

- https://wgilreath.github.io/ZeptoN/ZepC.java

The author and creator of ZeptoN welcomes feedback, comments, and questions!

About the Author

I am a developer, computer scientist, and writer with many years of development experience. I program in Java for work and fun.

Me, Myself, and I—Your Author's Picture

My home site about me is: **https://www.wfgilreath.xyz/**. I describe himself as: a writer of code, equations, poems, text, and lover of cats. He can be reached online at **will.f.gilreath@gmail.com** by e-mail.

Copyright

License

Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

- No additional restrictions – You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices:

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

Online:

This summary license and the full license is available online, respectively at:

- https://creativecommons.org/licenses/by-sa/4.0/
- https://creativecommons.org/licenses/by-sa/4.0/legalcode

Credits

- The first page photo by Kevin Ku from Pexels. Downloaded April 8, 2020.
https://www.pexels.com/photo/coding-computer-data-depth-of-field-577585/