JavaSpaced

Version 0.1.1

James Vetro

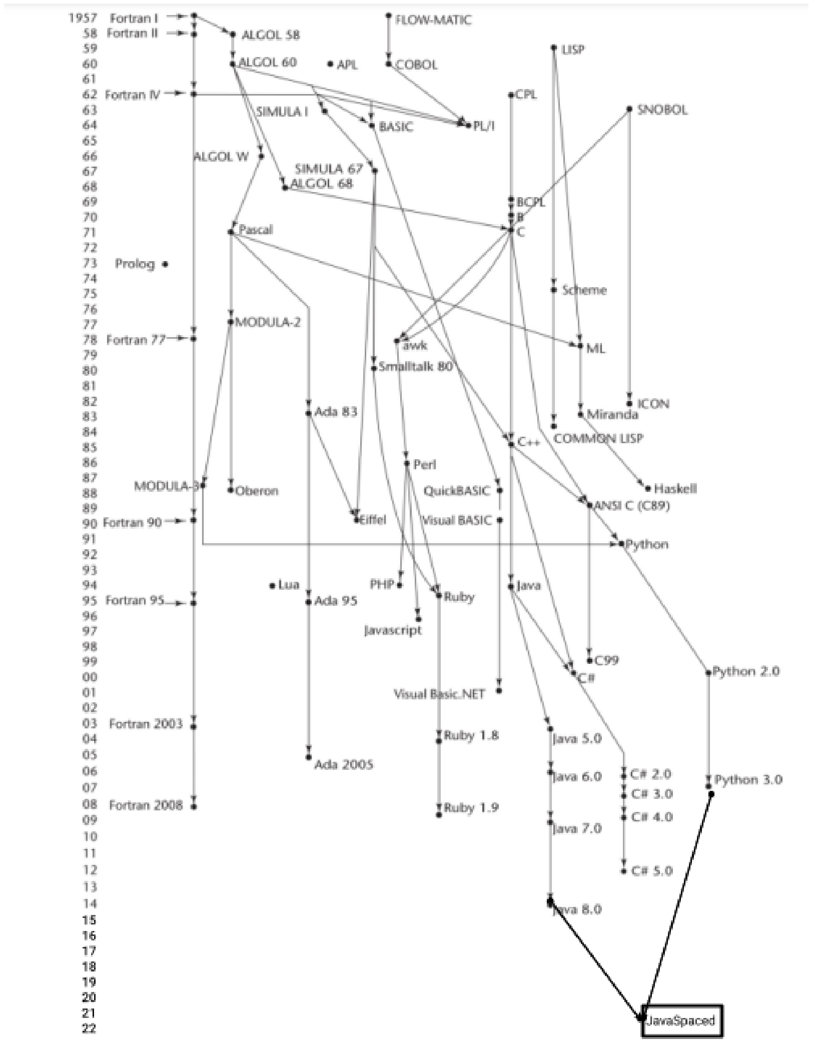
Table of Contents

1. Introduction

JavaSpaced is a new programming language, designed by James Vetro, as a modified version of Java, taking inspiration from Python. This language was named due to the initial goal of this language: To make Java formatting not just a nice dolled up thing, but a requirement. The new features of JavaSpaced include:

1. Much of the basic Java syntax and keywords, carried over into this new language
2. Whitespace requirements, removing the necessary semicolon, but leaving the option like Python
3. Some shortcuts as used in python (Print for system.out.print,).
4. Continued compatibility with all systems that are compatible with Java.
5. A compiled, static typed form, like Java.
6. New Read and Write shortcuts for files, as used in Python.
7. Parentheses and bracket use, for more obvious distinctions in loops and functions.
8. Java library compatibility, to allow for more diverse programming and some interchangability.
9. Compressed Float and Double, leaving double for more precise data, and int as normal.

1.1 Geneology

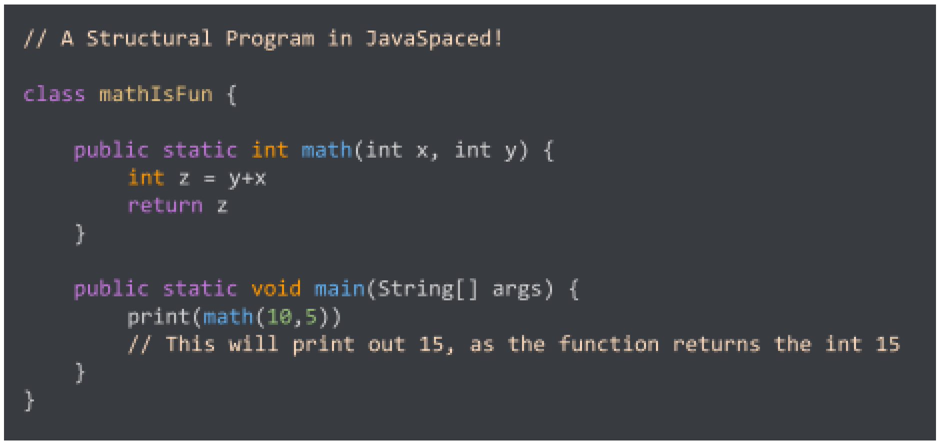


1.2 Hello World

```
// A Standard Hello World Program in JavaSpaced!

class HelloWorld {
    public static void main(String[] args) {
        print("Hello, World!")
    }
}
```

1.3 Program Structure



```
// A Structural Program in JavaSpaced!

class mathIsFun {

    public static int math(int x, int y) {
        int z = y+x
        return z
    }

    public static void main(String[] args) {
        print(math(10,5))
        // This will print out 15, as the function returns the int 15
    }
}
```

1.4 Types and Variables

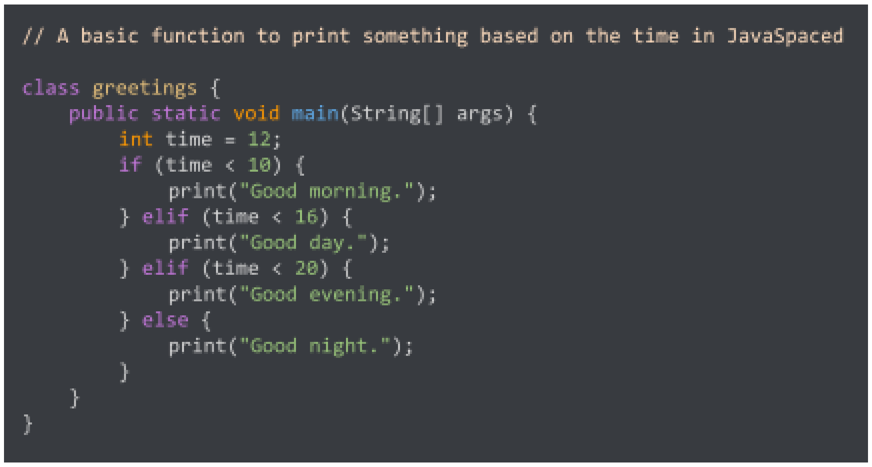
There are two kinds of types in JavaSpaced: value types and reference types. Variables of value types directly contain their data whereas variables of reference types store references to their data, the latter being known as objects. With reference types, it is possible for two variables to reference the same object and thus possible for operations on one variable to affect the object referenced by the other variable.

1.5 Visibility

Visibility in JavaSpaced is handled quite easily, because it follows the Java convention of determining visibility in the method declaration, such as public static void main, where public is the visibility of the method. JavaSpaced uses the same three visibility controls as Java, with Public, Protected, and Private.

1.6 Statements Differing from Java and Python

Statements	Examples

If, else if, else	
	```
	// A basic function to print something based on the time in JavaSpaced

	class greetings {
	    public static void main(String[] args) {
	        int time = 12;
	        if (time < 10) {
	            print("Good morning.");
	        } elif (time < 16) {
	            print("Good day.");
	        } elif (time < 20) {
	            print("Good evening.");
	        } else {
	            print("Good night.");
	        }
	    }
	}
	```

Print statement	Follows Python syntax not Java, with print, as opposed to system.out.print
Method declaration	Follows Java syntax of (as an example) public static void rather than just def

2. Lexical Structure

2.1 Programs

A JavaSpaced program consists of one or more source files. A source file is an ordered sequence of (probably Unicode) characters.
Conceptually speaking, a program is compiled using three steps:

1. Transformation, which converts a file from a particular character repertoire and encoding scheme into a sequence of Unicode characters.
2. Lexical analysis, which translates a stream of Unicode input characters into a stream of tokens.
3. Syntactic analysis, which translates the stream of tokens into executable code.

2.2 Grammars

This section is designed to show the syntax of JavaSpaced where it differs from Java and Python.

2.2.1 Lexical grammar where different from Java and Python

<assignmentOperator> → =
<arithmeticOperator> → + | * | / | -
<booleanOperator> → == | != | <= | >= | < | >
<print> → print

2.2.2 Syntactic (“parse”) grammar where different from Java and Python

<program> → class <string>{ <function>}

<function> → <visibility> <interactivity> <returnValue> <string>(<input>) {
<storage><procedure>} <function>

<declaration> → <type> <string> = <value>

<ifElseStatement> → case(<expr>) { <ifBool>} case2(<expr>) { <elifBool>}

2.3 Lexical Analysis

2.3.1 Comments

JavaSpaced offers single-line and delimited comments, following the syntax of Java. Single line comments start with // and go to the end of the source line. Delimited comments begin with /* and end with*/.

2.4 Tokens

There are several kinds of tokens: identifiers, keywords, literals, operators, and punctuators. Comments are not tokens, though they act as separators for tokens where needed. Whitespace IS a token in this language, unlike Java, given that it punctuates, replacing the ; token in Java.

Tokens:
identifier
keyword
integer-literal
real-literal
character-literal
string-literal
operator-or-punctuator

2.4.1 Keywords Different From Java

New Keywords: print, elif, in

Removed Keywords: system.out.print, else if

3. Type System

JavaSpaced is a strong static typed language, following Java’s basis. Strong typing means that type errors are caught and expressed to the programmer during compilation. Static typing means early binding compile-time type checking.

3.1 Type Rules

The type rules for JavaSpaced are as follows:

Assignment:

⊢ e1 : T
⊢ e2 : T
T is a primitive type

⊢ e1 = e2 : T

Comparisons

⊢ e1 : T
⊢ e2 : T
T is a primitive type

⊢ e1 == e2 : boolean,
OR
⊢ e1 >= e2 : boolean,
OR
⊢ e1 <= e2 : boolean,
OR
⊢ e1 > e2 : boolean,
OR
⊢ e1 < e2 : boolean,
OR
⊢ e1 != e2 : boolean,

Addition

S ⊢ e1 : T
S ⊢ e2 : T
T is a primitive type

S ⊢ e1 + e2 : T

3.2 Value types

Name	Size	Minimum	Maximum
Boolean	1 bit	False	True
Byte	8 bits	-128	127
Char	16 bits	-128	127
Short	16 bits	-2^{15} (-32,768)	$2^{15}-1$ (-32,767)
Int	32 bits	-2^{31} (-2,147,483,648)	$2^{31}-1$ (-2,147,483,647)
Double	64 bits	$4.39x10^{-326}$	$1.79x10^{+308}$
Long	64 bits	-2^{63}	$2^{63}-1$

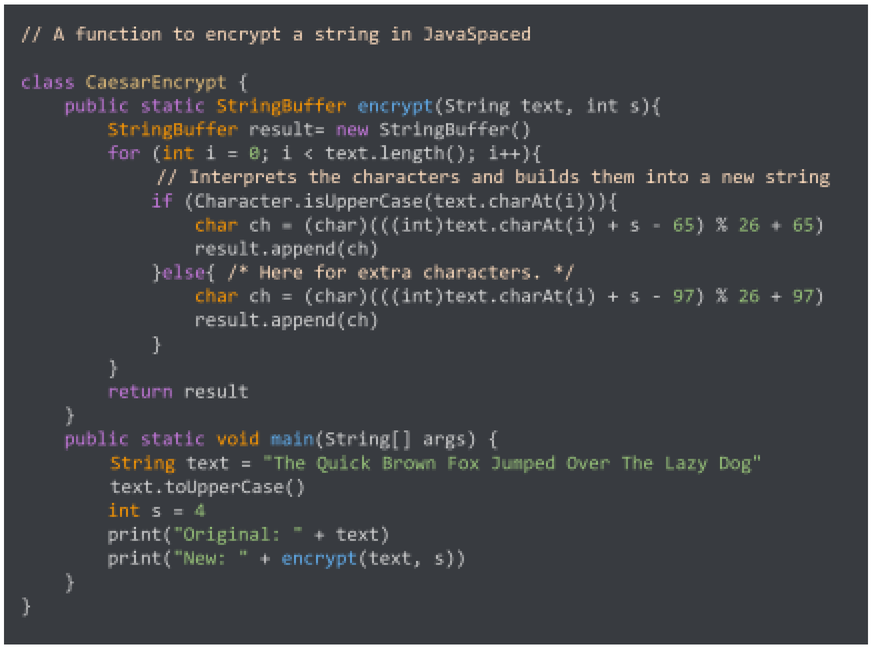
3.3 Reference Types

Array: an immutable collection of data with a defined length.
String: an array of Char
Object: a collection of code which can be run as a function.

4. Example Programs

Test programs written in JavaSpaced, to demonstrate the functionality of this new language!

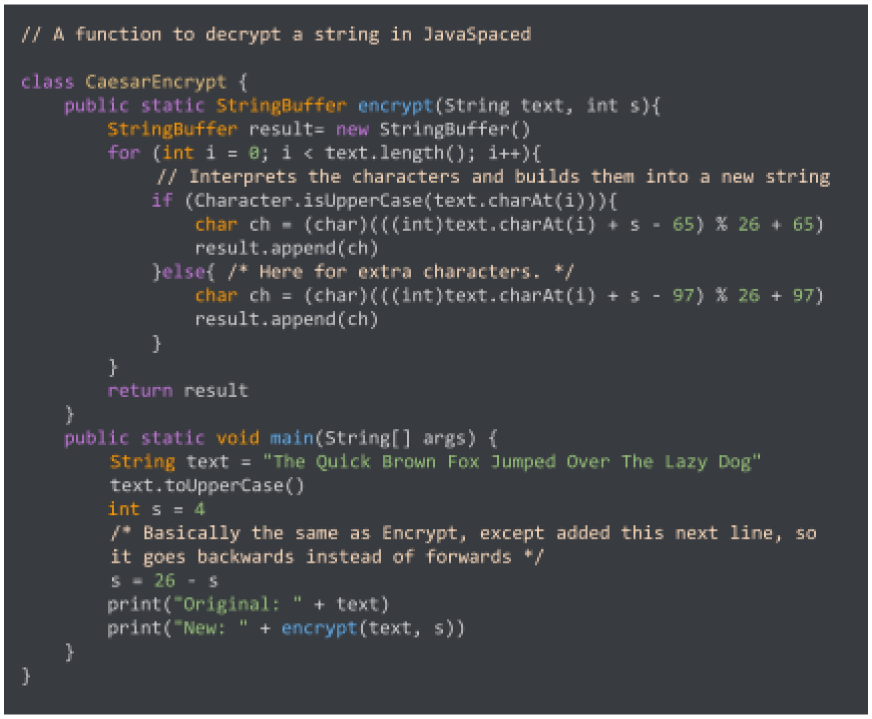
2.1 Caesar Encrypt



```
// A function to encrypt a string in JavaSpaced

class CaesarEncrypt {
    public static StringBuffer encrypt(String text, int s){
        StringBuffer result= new StringBuffer()
        for (int i = 0; i < text.length(); i++){
            // Interprets the characters and builds them into a new string
            if (Character.isUpperCase(text.charAt(i))){
                char ch = (char)(((int)text.charAt(i) + s - 65) % 26 + 65)
                result.append(ch)
            }else{ /* Here for extra characters. */
                char ch = (char)(((int)text.charAt(i) + s - 97) % 26 + 97)
                result.append(ch)
            }
        }
        return result
    }
    public static void main(String[] args) {
        String text = "The Quick Brown Fox Jumped Over The Lazy Dog"
        text.toUpperCase()
        int s = 4
        print("Original: " + text)
        print("New: " + encrypt(text, s))
    }
}
```

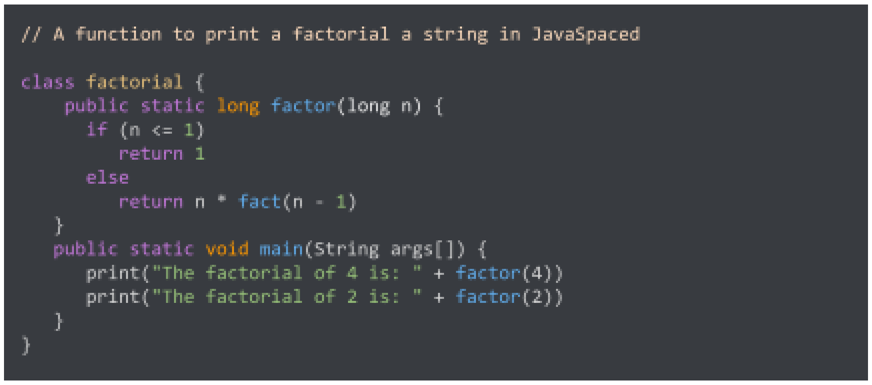
2.2 Caesar Decrypt



```
// A function to decrypt a string in JavaSpaced

class CaesarEncrypt {
    public static StringBuffer encrypt(String text, int s){
        StringBuffer result= new StringBuffer()
        for (int i = 0; i < text.length(); i++){
            // Interprets the characters and builds them into a new string
            if (Character.isUpperCase(text.charAt(i))){
                char ch = (char)(((int)text.charAt(i) + s - 65) % 26 + 65)
                result.append(ch)
            }else{ /* Here for extra characters. */
                char ch = (char)(((int)text.charAt(i) + s - 97) % 26 + 97)
                result.append(ch)
            }
        }
        return result
    }
    public static void main(String[] args) {
        String text = "The Quick Brown Fox Jumped Over The Lazy Dog"
        text.toUpperCase()
        int s = 4
        /* Basically the same as Encrypt, except added this next line, so
        it goes backwards instead of forwards */
        s = 26 - s
        print("Original: " + text)
        print("New: " + encrypt(text, s))
    }
}
```

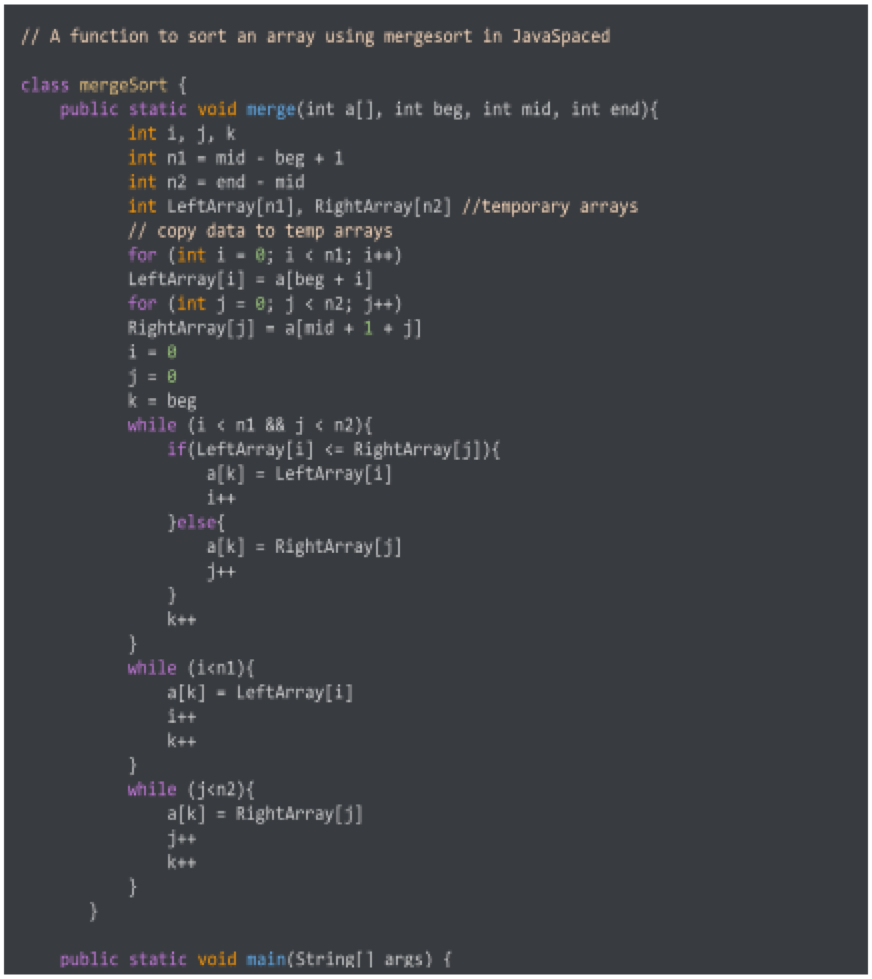
2.3 Factorial



```
// A function to print a factorial a string in JavaSpaced

class factorial {
    public static long factor(long n) {
      if (n <= 1)
         return 1
      else
         return n * fact(n - 1)
   }
   public static void main(String args[]) {
      print("The factorial of 4 is: " + factor(4))
      print("The factorial of 2 is: " + factor(2))
   }
}
```

2.4 Merge Sort

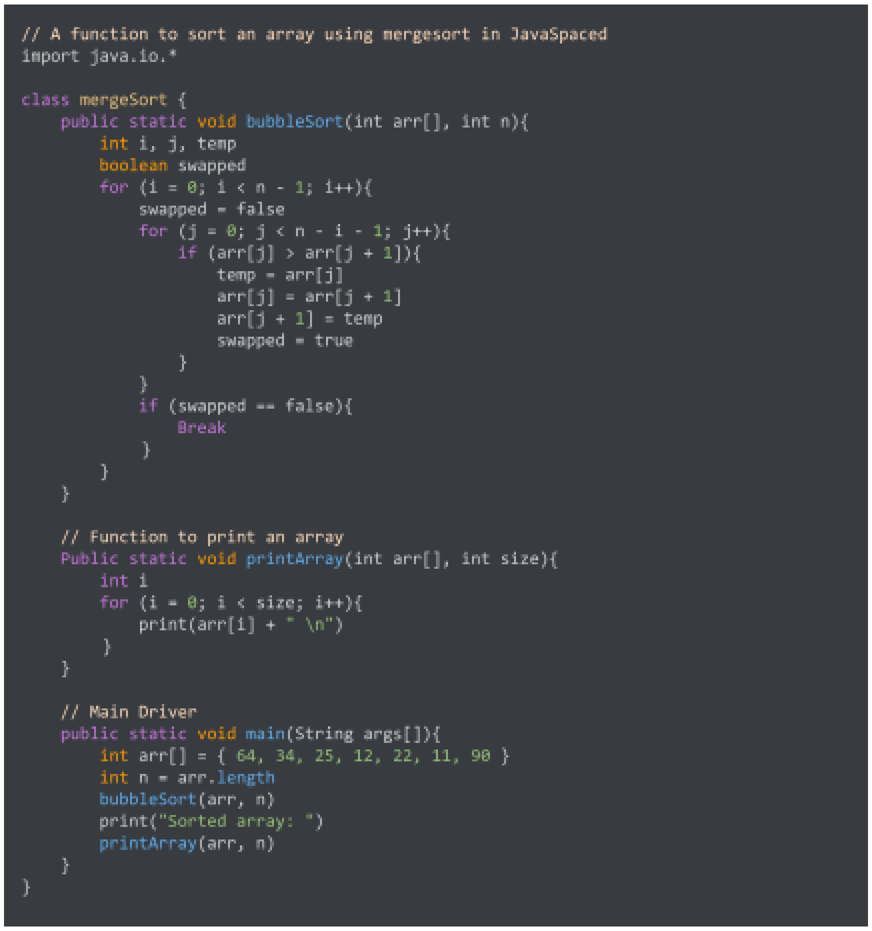


```
// A function to sort an array using mergesort in JavaSpaced

class mergeSort {
    public static void merge(int a[], int beg, int mid, int end){
           int i, j, k
           int n1 = mid - beg + 1
           int n2 = end - mid
           int LeftArray[n1], RightArray[n2] //temporary arrays
           // copy data to temp arrays
           for (int i = 0; i < n1; i++)
           LeftArray[i] = a[beg + i]
           for (int j = 0; j < n2; j++)
           RightArray[j] = a[mid + 1 + j]
           i = 0
           j = 0
           k = beg
           while (i < n1 && j < n2){
               if(LeftArray[i] <= RightArray[j]){
                   a[k] = LeftArray[i]
                   i++
               }else{
                   a[k] = RightArray[j]
                   j++
               }
               k++
           }
           while (i<n1){
               a[k] = LeftArray[i]
               i++
               k++
           }
           while (j<n2){
               a[k] = RightArray[j]
               j++
               k++
           }
       }

    public static void main(String[] args) {
```

2.5 Bubble Sort



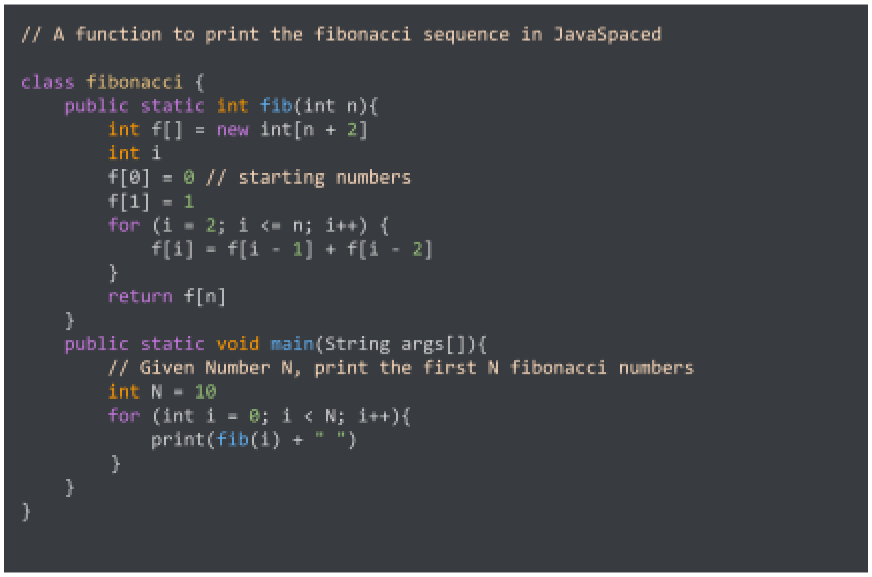
```
// A function to sort an array using mergesort in JavaSpaced
import java.io.*

class mergeSort {
    public static void bubbleSort(int arr[], int n){
        int i, j, temp
        boolean swapped
        for (i = 0; i < n - 1; i++){
            swapped = false
            for (j = 0; j < n - i - 1; j++){
                if (arr[j] > arr[j + 1]){
                    temp = arr[j]
                    arr[j] = arr[j + 1]
                    arr[j + 1] = temp
                    swapped = true
                }
            }
            if (swapped == false){
                Break
            }
        }
    }

    // Function to print an array
    Public static void printArray(int arr[], int size){
        int i
        for (i = 0; i < size; i++){
            print(arr[i] + " \n")
        }
    }

    // Main Driver
    public static void main(String args[]){
        int arr[] = { 64, 34, 25, 12, 22, 11, 90 }
        int n = arr.length
        bubbleSort(arr, n)
        print("Sorted array: ")
        printArray(arr, n)
    }
}
```

2.6 Fibbonacci Sequence



```
// A function to print the fibonacci sequence in JavaSpaced

class fibonacci {
    public static int fib(int n){
        int f[] = new int[n + 2]
        int i
        f[0] = 0 // starting numbers
        f[1] = 1
        for (i = 2; i <= n; i++) {
            f[i] = f[i - 1] + f[i - 2]
        }
        return f[n]
    }
    public static void main(String args[]){
        // Given Number N, print the first N fibonacci numbers
        int N = 10
        for (int i = 0; i < N; i++){
            print(fib(i) + " ")
        }
    }
}
```