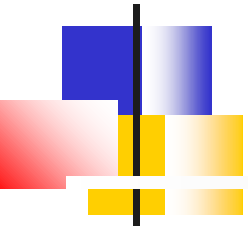
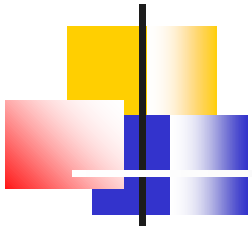


ooeForth 2.04

SVFIG

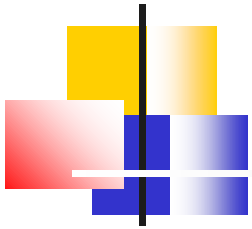
Chen-Hanson Ting

July 24, 2021



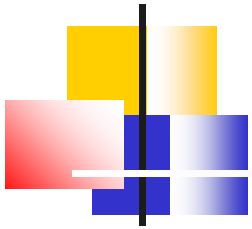
Authors

- Xuyang (Shawn) Chen
- Brad Nelson
- Chochain Lee
- Chen-Hanson Ting



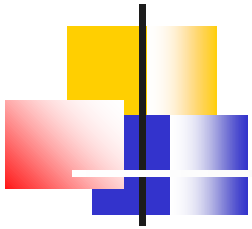
ooeForth

- I wanted a simple Forth in Java modeled after jeforth614.
- Forth words, integer literals, string literals, and address literals should all be objects.
- All colon words should have linear object lists.
- A true object-oriented Forth.



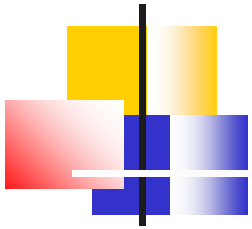
ooeEforth 2.04

- There are only two types of words:
  - Primitive words
  - Colon words
- All system words are primitive objects.
- All user defined words are colon objects.



ooeForth 2.04

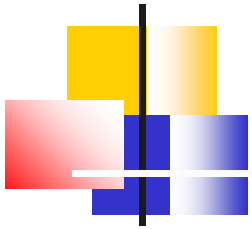
- A single Class Code constructs all Forth words as objects.
- All primitive objects are stored in a dictionary using their names as keys.
- Colon objects are sequentially added to the dictionary.



ooeForth 2.04

- All colon objects contain linear object lists.
- All colon objects are executed by this very simple inner interpreter:

```
nest(){for(var w:pf) w.xt();}
```



Eforth204 Class

- Stack
- Rstack
- Dictionary
  - Primitive objects + Colon objects
- Methods
  - main(), setup)dictionary(), outerInterpreter()
- Class ForthList manages the dictionary
- Class Code constructs all objects

class Eforth204

stack

rstack

dictionary

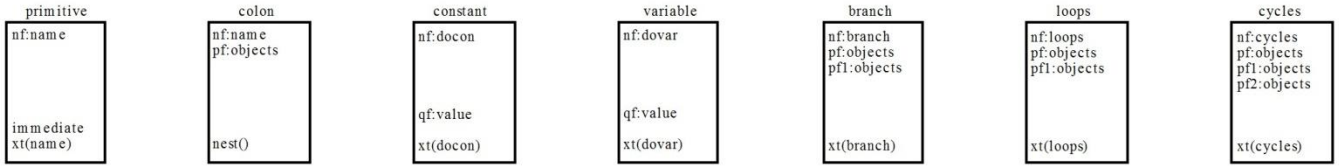
compiling

base = 10;

fence = 0;

wp,ip;

idiom;



Class Eforth204

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.time.LocalTime;
import java.util.function.*;
public class Eforth203 {    // ooeforth 2.04
  static Scanner in;
  static Stack<Integer> stack=new Stack<>();
  static Stack<Integer> rstack=new Stack<>();
  static ForthList<Code> dictionary=new ForthList<>();
  static boolean compiling=false;
  static int base=10; static int wp,ip;
  static Frame frame = new Frame("ooeForth");
  static TextArea input = new TextArea("words",10,50);
  static TextArea output = new TextArea("ooeForth 2.04\n",10,80);
  static void setup_dictionary() {
     lookUp.forEach((k,v)->dictionary.add(new Code(k))); // create primitive words
     final String immd[]= {
              ";",".(","$\"","\\","."," .\"",
              "aft","again","begin","else","for","if",
              "next","repeat","then","until","while"};
     for (String s: immd) {
        dictionary.find(s,w->s.equals(w.name)).immediate(); // set immediate flag
     }}
  public static void main(String args[]) {
     System.out.println("ooeForth2.03\n");
     setup_dictionary();
     // GetKeyChar
     Font font= new Font("Monospaced", Font.PLAIN, 12);
     input.setFont(font);
     output.setFont(font);
     frame.add(input, BorderLayout.EAST);
     frame.add(output, BorderLayout.WEST);
     frame.setSize(1000, 700);
     frame.setVisible(true);
     frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent we) {
           System.exit(0);            }});
     input.addKeyListener(new KeyAdapter() {
        public void keyTyped(KeyEvent ke) {
           char keyChar = ke.getKeyChar();
           if (keyChar <= 13) {
              in = new Scanner(input.getText());
              outerInterpreter();
              for(int n:stack) System.out.print(Integer.toString(n,base)+" ");
              System.out.print(">ok\n");
              input.setText("");
              in.close();
     }}});    }
```

Forth Outer Interpreter

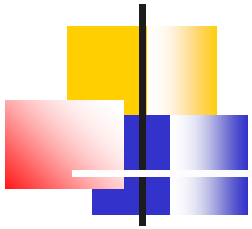
```
public static void outerInterpreter() {              // ooeforth 2.01
   while(in.hasNext()) {          // parse input
      String idiom=in.next();
      Code newWordObject=dictionary.find(idiom,w->idiom.equals(w.name));
      if(newWordObject !=null) {          // word found
         if(!compiling) || newWordObject.immediate) {
            try {newWordObject.xt(); }              // execute
            catch (Exception e) {output.append(e.toString());}}
        else {        // or compile
            dictionary.tail().addCode(newWordObject);}}
      else {
         try {int n=Integer.parseInt(idiom, base);            // not word, try number
         if (compiling) {                    // compile integer literal
            dictionary.tail().addCode(new Code("dolit",n));}
         else { stack.push(n);}}                    // or push number on stack
         catch (NumberFormatException  ex) {  // catch number errors
            output.append(idiom + "? ");
            compiling=false; stack.clear();}
      }}
   for(int n:stack) output.append(Integer.toString(n,base)+" ");
   output.append(">ok\n");
}
```

Class Code

```
   static class Code {                    // one size fits all objects
      static int fence=0; public int token=0; public String name;
      public ForthList<Code> pf=new ForthList<>();
      public ForthList<Code> pf1=new ForthList<>();
      public ForthList<Code> pf2=new ForthList<>();
      public ForthList<Integer> qf=new ForthList<>() ;
      public int struct=0; public boolean immediate=false;
      public String literal;
      public Code(String n) {name=n;token=fence++;}
      public Code(String n, boolean f) {name=n; if (f) token=fence++;}
      public Code(String n, int d) {name=n;qf.add(d);}
      public Code(String n, String l) {name=n;literal=l;}
      public Code immediate() { immediate=true; return this; }
      public void xt() {
         if (lookUp.containsKey(name)) {
            lookUp.get(name).accept(this);                  // run primitives words
         } else { rstack.push(wp); rstack.push(ip);          // run colon words
         wp=token; ip=0;
         for(Code w:pf) {
            try { w.xt();ip++;}                             // inner interpreter
            catch (ArithmeticException e) {}}
         ip=rstack.pop(); wp=rstack.pop();}}
      public Code addCode(Code w) {this.pf.add(w);return this;}
   }
```

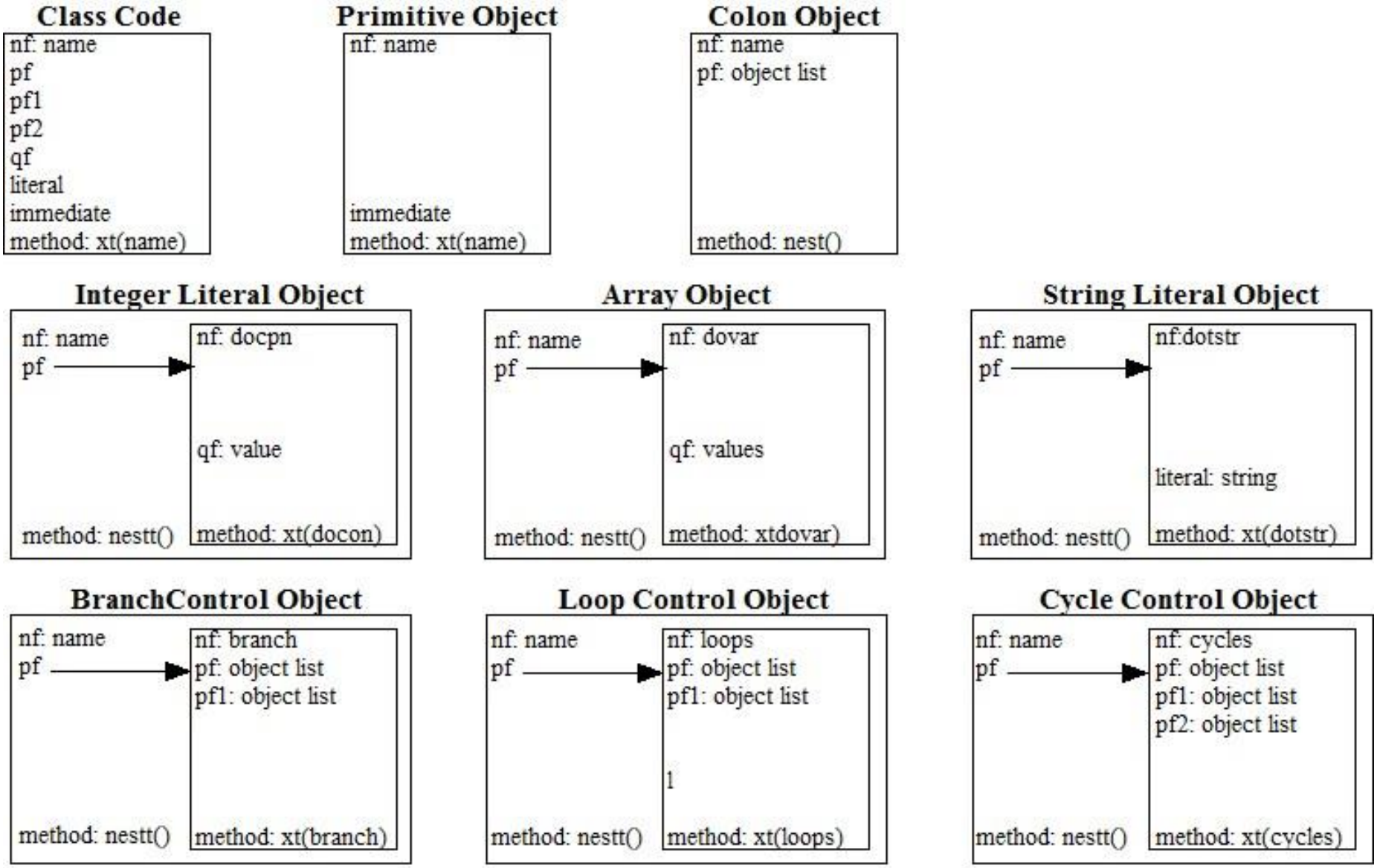
Primitive Methods

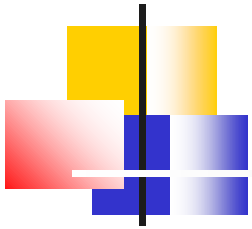
```
   static public HashMap<String, Consumer<Code>> lookUp=new HashMap<>() {{
      // stacks
      put("dup",c->stack.push(stack.peek()));
      put("over",c->stack.push(stack.get(stack.size()-2)));
      put("2dup",c->stack.addAll(stack.subList(stack.size()-2,stack.size())));
      put("2over",c->stack.addAll(stack.subList(stack.size()-4,stack.size()-2)));
      put("4dup",c->stack.addAll(stack.subList(stack.size()-4,stack.size())));
      put("swap",c->stack.add(stack.size()-2,stack.pop()));
      put("rot",c->stack.push(stack.remove(stack.size()-3)));
      put("-rot",c->{stack.push(stack.remove(stack.size()-3));stack.push(stack.remove(stack.size()-3)
                                                                        );});
      put("2swap",c->{stack.push(stack.remove(stack.size()-4));stack.push(stack.remove(stack.size(
                                                                        )-4));});
      put("pick",c->{int i=stack.pop();int n=stack.get(stack.size()-i-1);stack.push(n);});
      put("roll",c->{int i=stack.pop();int n=stack.remove(stack.size()-i-1);stack.push(n);});
      put("drop",c->stack.pop());
      put("nip",c->stack.remove(stack.size()-2));
      put("2drop",c->{stack.pop();stack.pop();});
      put(">r",c->rstack.push(stack.pop()));
      put("r>",c->stack.push(rstack.pop()));
      put("r@",c->stack.push(rstack.peek()));
      put("push",c->rstack.push(stack.pop()));
      put("pop",c->stack.push(rstack.pop()));
      // math
      put("+",c->stack.push(stack.pop()+stack.pop()));
      put("-",c->{int n=stack.pop();stack.push(stack.pop()-n);});
      put("*",c->stack.push(stack.pop()*stack.pop()));
      put("/",c->{int n=stack.pop();stack.push(stack.pop()/n);});
      put("*/",c->{int n=stack.pop();stack.push(stack.pop()*stack.pop()/n);});
      put("*/mod",c->{int n=stack.pop();int m=stack.pop()*stack.pop();
        stack.push(m%n);stack.push(m/n);});
      put("mod",c->{int n=stack.pop();stack.push(stack.pop()%n);});
      put("and",c->stack.push(stack.pop()&stack.pop()));
      put("or",c->stack.push(stack.pop()|stack.pop()));
      put("xor",c->stack.push(stack.pop()^stack.pop()));
      put("negate",c->stack.push(-stack.pop()));
      // logic                    // point to current object
      put("0=",c->stack.push((stack.pop()==0)?-1:0));
      put("0<",c->stack.push((stack.pop()<0)?-1:0));
      put("0>",c->stack.push((stack.pop()>0)?-1:0));
      put("=",c->{int n=stack.pop();stack.push((stack.pop()==n)?-1:0);});
      put(">",c->{int n=stack.pop();stack.push((stack.pop()>n)?-1:0);});
      put("<",c->{int n=stack.pop();stack.push((stack.pop()<n)?-1:0);});
      put("<>",c->{int n=stack.pop();stack.push((stack.pop()!=n)?-1:0);});
      put(">=",c->{int n=stack.pop();stack.push((stack.pop()>=n)?-1:0);});
      put("<=",c->{int n=stack.pop();stack.push((stack.pop()<=n)?-1:0);});
      // output
      put("base@",c->stack.push(base));
```



Forth Objects

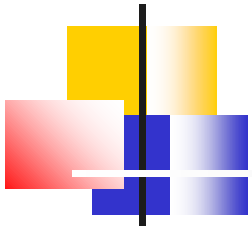
- Primitive objects
- Colon Objects
  - Integer literal objects
  - String literal objects
  - Control structure objects
    - Branch objects
    - Loops objects
    - Cycles objects


Class Code
nf: name
pf
pf1
pf2
qf
literal
immediate
method: xt(name)
Primitive Object
nf: name
immediate
method: xt(name)
Colon Object
nf: name
pf: object list
method: nest()
Integer Literal Object
nf: name
pf
nf: docpn
qf: value
method: nestt()
method: xt(docon)
Array Object
nf: name
pf
nf: dovar
qf: values
method: nestt()
method: xtdovar)
String Literal Object
nf: name
pf
nf:dotstr
literal: string
method: nestt()
method: xt(dotstr)
BranchControl Object
nf: name
pf
nf: branch
pf: object list
pf1: object list
method: nestt()
method: xt(branch)
Loop Control Object
nf: name
pf
nf: loops
pf: object list
pf1: object list
1
method: nestt()
method: xt(loops)
Cycle Control Object
nf: name
pf
nf: cycles
pf: object list
pf1: object list
pf2: object list
method: nestt()
method: xt(cycles)



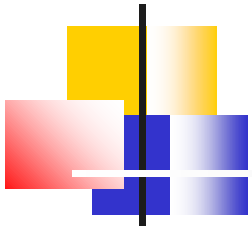
Primitive Object

- **nf: name**
- **token: id**
- pf
- pf1
- pf2
- qf
- **immediate: flag**
- **method: `xt(name)`**



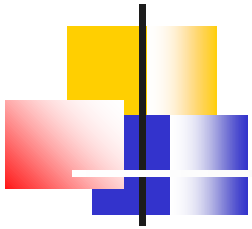
Colon Object

- **nf: name**
- **token: id**
- **pf: object list**
- pf1
- pf2
- qf
- immediate
- **method:** `nest()`



Integer Object

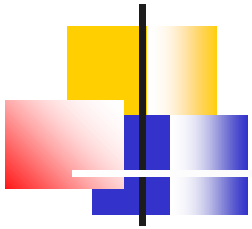
- **nf: docon**
- **token: id**
- pf:
- pf1
- pf2
- **qf: value**
- immediate
- **method:** `xt(docon)`



String Object

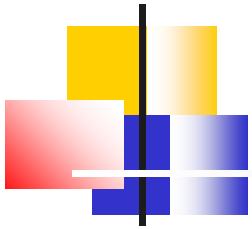
- **nf: dostr[dotstr]**
- **token: id**
- pf:
- pf1
- pf2
- **literal: string**
- immediate
- **method:** `xt(dostr[dotstr])`

```
Usage: $" xxx" , ." yyy"
```



Control Structures

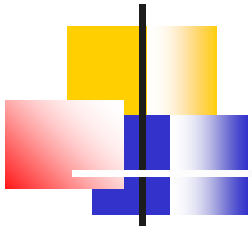
- There are branches and loops in an object list.
- 5 control structures are colon objects with alternate paths:
  - `if pf else pf1 then`
  - `begin pf again`
  - `begin pf until`
  - `begin pf while pf1 repeat`
  - `for pf aft pf1 then pf2 next`



Branch Object

- **nf: branch**
- **token: id**
- **pf: object list**
- **pf1: object list**
- pf2
- qf
- immediate
- **method:** `xt(branch)`

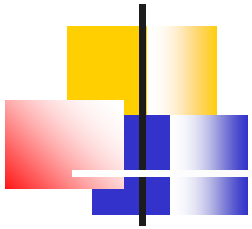
```
Usage: if pf else pf1 then
```



Loops Object

- **nf: loops**
- **token: id**
- **pf: object list**
- **pf1: object list**
- pf2
- qf
- immediate
- **method:** `xt(loops)`

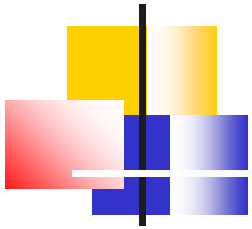
`Usage: begin pf while pf1 repeat`



Cycles Object

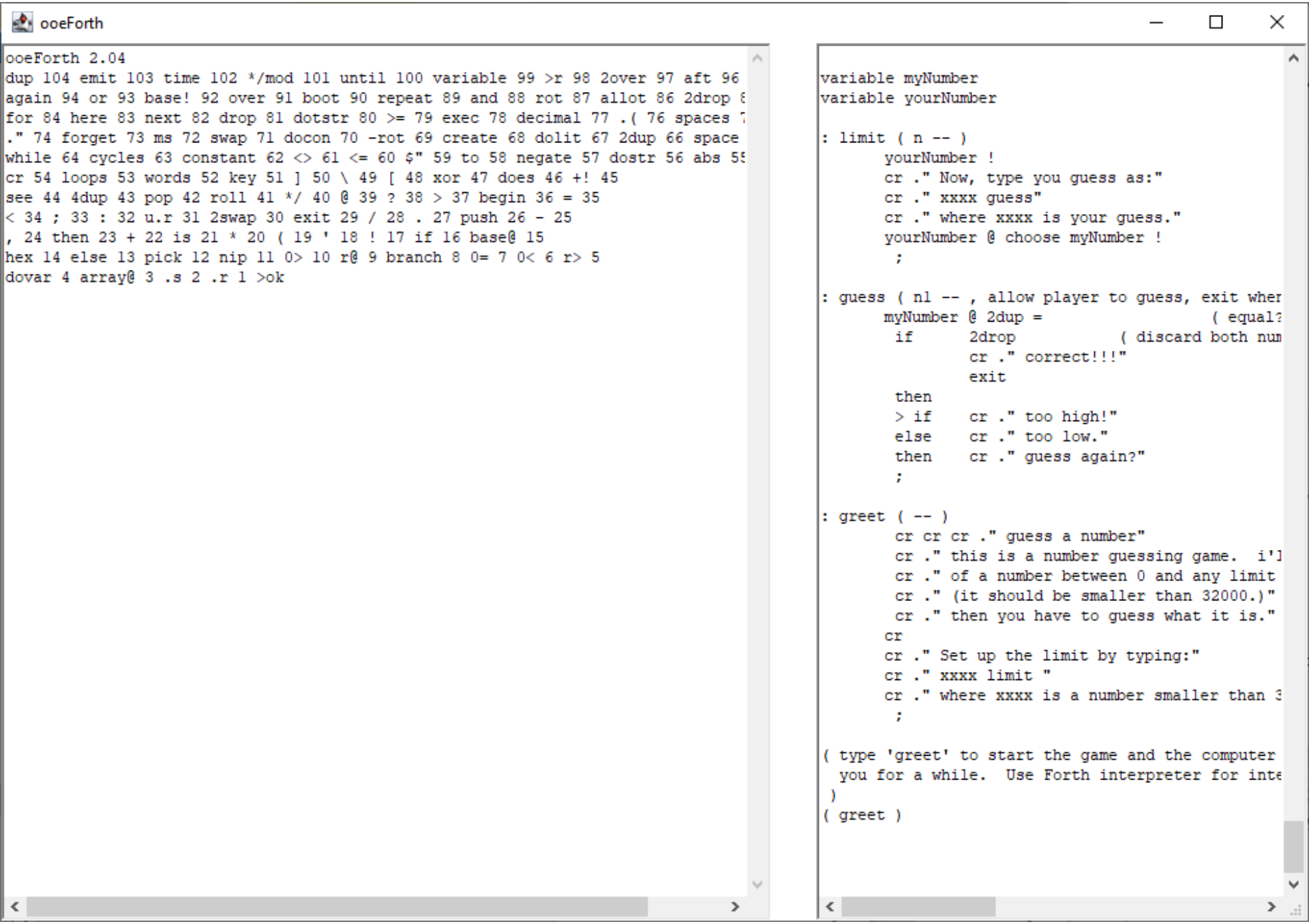
- **nf: cycles**
- **token: id**
- **pf: object list**
- **pf1: object list**
- **pf2: object list**
- qf
- immediate
- **method:** `xt(cycles)`

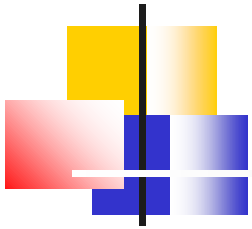
Usage: `for pf aft pf1 then pf2 next`



Windows Interface

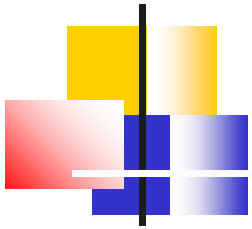
- **Methods in java.awt class are used to built the Windows interface:**
  - **Frame `frame` is a container**
  - **TextArea `input` receives text input**
  - **TextArea `output` displays output text**


ooeForth
ooeForth 2.04
dup 104 emit 103 time 102 */mod 101 until 100 variable 99 >r 98 2over 97 aft 96
again 94 or 93 base! 92 over 91 boot 90 repeat 89 and 88 rot 87 allot 86 2drop 8
for 84 here 83 next 82 drop 81 dotstr 80 >= 79 exec 78 decimal 77 .(76 spaces 7
." 74 forget 73 ms 72 swap 71 docon 70 -rot 69 create 68 dolit 67 2dup 66 space
while 64 cycles 63 constant 62 <> 61 <= 60 $" 59 to 58 negate 57 dostr 56 abs 55
cr 54 loops 53 words 52 key 51] 50 \ 49 [48 xor 47 does 46 +! 45
see 44 4dup 43 pop 42 roll 41 */ 40 @ 39 ? 38 > 37 begin 36 = 35
< 34 ; 33 : 32 u.r 31 2swap 30 exit 29 / 28 . 27 push 26 - 25
, 24 then 23 + 22 is 21 * 20 (19 ' 18 ! 17 if 16 base@ 15
hex 14 else 13 pick 12 nip 11 0> 10 r@ 9 branch 8 0= 7 0< 6 r> 5
dovar 4 array@ 3 .s 2 .r 1 >ok
variable myNumber
variable yourNumber
: limit (n --)
yourNumber !
cr ." Now, type you guess as:"
cr ." xxxx guess"
cr ." where xxxx is your guess."
yourNumber @ choose myNumber !
;
: guess (nl -- , allow player to guess, exit wher
myNumber @ 2dup = (equal?
if 2drop (discard both num
cr ." correct!!!"
exit
then
> if cr ." too high!"
else cr ." too low."
then cr ." guess again?"
;
: greet (--)
cr cr cr ." guess a number"
cr ." this is a number guessing game. i'l
cr ." of a number between 0 and any limit
cr ." (it should be smaller than 32000.)"
cr ." then you have to guess what it is."
cr
cr ." Set up the limit by typing:"
cr ." xxxx limit "
cr ." where xxxx is a number smaller than 3
;
(type 'greet' to start the game and the computer
you for a while. Use Forth interpreter for inte
)
(greet)



Outer Interpreter

- **The Forth `outerInterpreter()` is called by a key event when the return key is pressed to evaluate text entered in the `input` box.**
- **The parser is simply a Java method: `Scanner.in.next()`.**

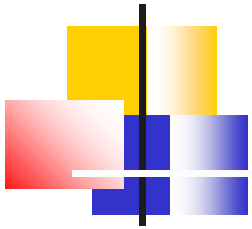


Inner Interpreter

- Object lists in colon objects are all linear object lists.
- Linear object lists can be executed by this very simple inner interpreter:

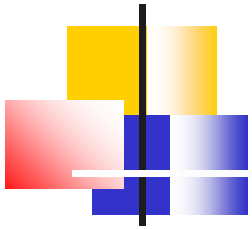
```
nest(){for(var w:pf) w.xt();}
```

- Linear object lists can be terminated early by throwing an `ArithmeticException`.



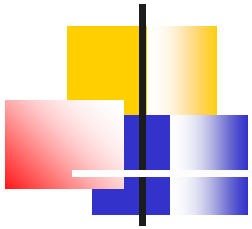
Linear Object Lists

- **Colon objects compile linear object lists in their `pf` fields.**
- **Linear lists can be nested indefinitely to solve complicated problems.**



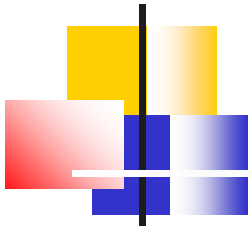
Conclusions

- Eforth204 implements Forth words as true objects.
- Eforth204 has only 405 lines of source code.
- It is the second smallest Forth after jeforth614 with 379 lines.
- It is the smallest Windows Forth ever.



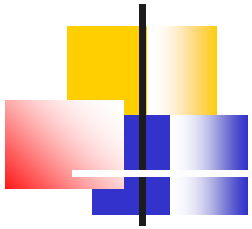
Windows Forth

- I documented figForth, polyForth, F83, and F-PC.
- I gave up on Win32Forth, because it was too complicated.
- For my own sanity, I started eForth.
- Cheahsin Yap gave me F# and I survived on it.



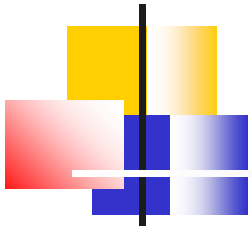
Windows Forth

- Windows might be complicated, but Windows interface should not.
- Eforth204 is simple because it uses only text input and output.
- If you got into files, menus, and buttons, things quickly became orders of magnitude more complicated.

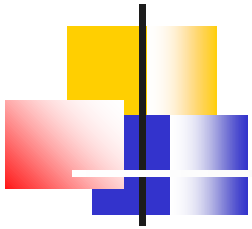


Link to Eforth204

- **Link to Eforth204:**
  - https://drive.google.com/file/d/1sOnnh8uez-9z1Gy9jHYHYhhCEOZEtFq1/view?usp=sharing
- **Email comments to me:**
- **chenhansunding@gmail.com**



Demo



Thank You!