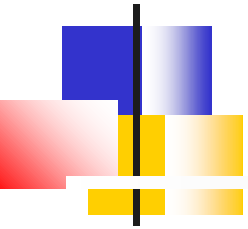
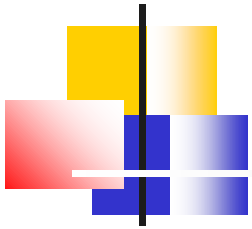


Java Eforth108

SVFIG

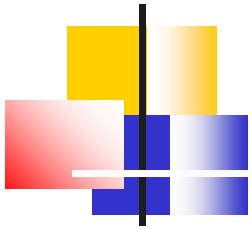
Chen-Hanson Ting

May 22, 2021



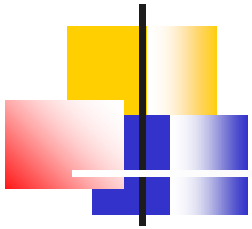
Java Forth

- There are several Forth implemented in Java.
- There were even an eForth implemented in Java by Michael A. Losh, in 1997.
- They are all very complicated.



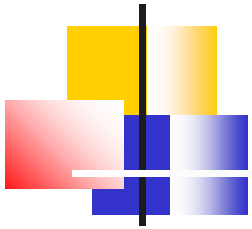
Java Eforth

- I wanted a simple Java Forth modeled after jeforth614.
- Every Forth word should be an object.
- To write and test Java code, you need an IDE, like Eclipse from IBM.



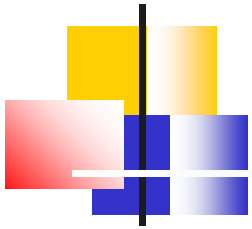
JavaScript

- Everybody knows that JavaScript is no Java.
- After learning some Java, I realized that JavaScript is actually a cScript.
- Objects in JavaScript are not objects, but very flexible arrays.



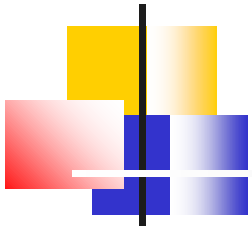
Eforth in Java

- Eforth101, experiments on stack.
- Eforth102, Brad Nelson made it a 4-function calculator..
- Eforth104, Shawn Chen made it a Forth.
- Eforth106, eForth prototype.
- Eforth108, aligned with forth614
- Eventually it will be released as javaEforth.



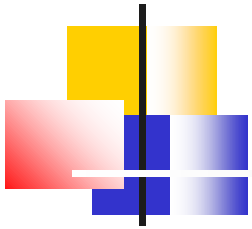
Java Eforth Objects

- **All Forth objects have the following attributes:**
  - `name`
  - `Token` **with an sequential ID**
  - `Pf` **with an object list**
  - `Qf` **with a data list**
  - `Literal` **with a string**
  - `Immediate` **with a compiling flag**



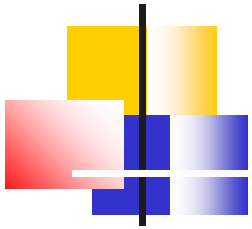
Eforth108

- A single method with a giant switch structure to execute all primitive words.
- The default method in this switch structure executes colon words.
- Constants, variables, and arrays are all implemented as colon words.



Eforth108

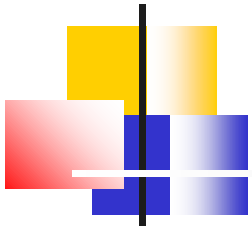
- Only two types of words:
  - Primitive words
  - Colon words
- All primitive words are constructed by Class Code.
- Colon words are defined by the user.



Outer Interpreter

- **The Forth outer interpreter is the `main()` method, written in Java.**
- **The parser is a single Java command: `Scanner.in.next()`.**
- **To use `in.next()`. I sacrificed the universal Forth prompt `OK`, and the opportunity to show the contents of data stack.**

```
in=new Scanner(System.in);String idiom;
while(!(idiom=in.next()).equals("bye")){
Code newWordObject=null;
  for (var w : dictionary){
    if (w.name.equals(idiom)) {newWordObject=w
    if(newWordObject != null){
      if((!compiling) || newWordObject.immedia
      else{  Code latestWord=dictionary.get(di
      latestWord.addWord(newWordObject);}}
    else{try {int n=Integer.parseInt(idiom, ba
      if (compiling){Code latestWord=dictionar
        latestWord.addWord(new Code("dolit",n)
      else{stack.push(n);}}
      catch (NumberFormatException  ex) {Syste
        compiling=false,stack.clear();}}}
  System.out.println("Thank you.");in.close();
```

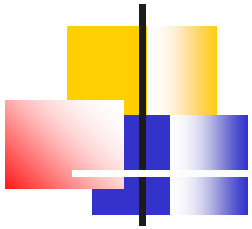


Inner Interpreter

- **Shawn Chen contributed a beautifully simple inner interpreter to process nested token lists:**

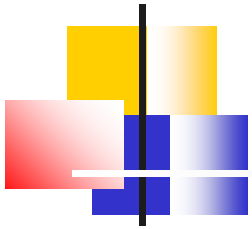
  ```
  for(Code w:words) w.xt();
  ```

- **I have to break up this code to handle control structures.**



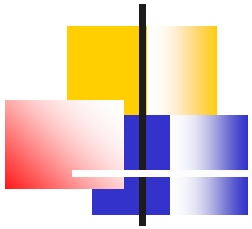
Inner Interpreter

```
default: {
rstack.push(wp);rstack.push(ip);
wp=token;ip=0;
while(!pf.get(ip).name.equals("exit")) {
     try{pf.get(ip).xt();  }
     catch (Exception e)
     {System.out.println(e);}
     ip++;
     }
ip=rstack.pop();wp=rstack.pop();
}
```



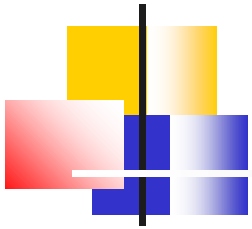
Object List

- Each word object has a sequential ID as a token.
- However, colon words compile object lists, not token lists.
- Tokens are still needed to look up objects in dictionary.



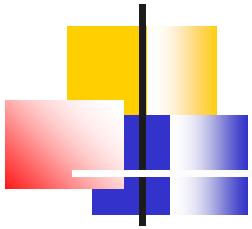
Literals in Object List

- In a colon word, the `pf` attribute contains an object list.
- Forth needs these literals in the object list:
  - Integer literals
  - String literals
  - Address literals



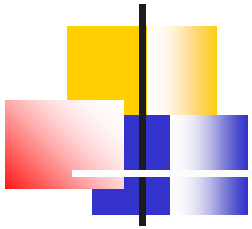
Literals in Object List

- **Object `dolit` contains an integer in its `qf`.**
- **Objects `dostr` and `dotstr` contain strings in their `literal`.**
- **Objects `branch`, `zbranch`, and `donext` have the branching addresses in their `qf`.**



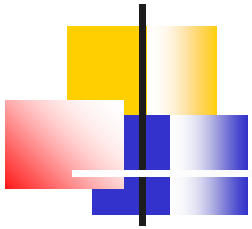
Lessons108.txt

- 17 lessons/tutorials on how to use eForth were used to verify that Eforth108 worked properly.
- The entire file can be copied and pasted into the Console window to exercise Eforth108.



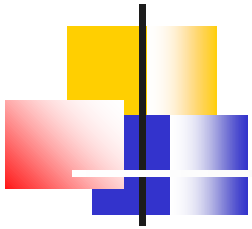
Conclusions

- **Eforth108 proves that Forth words work as true objects.**
- **Eforth108 is logically correct but can use lots of improvement.**
- **It is my first Java project and shows my lack of understanding of this extremely complicated language.**

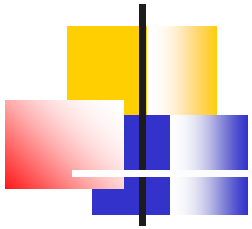


Link to Eforth108

- **Link to Eforth108:**
  - **https://drive.google.com/file/d/1S-UpZ73k3mPx9zU7J_dcP49kRQ4GPI9M/view?usp=sharing**
- **Email for commenting:**
  - **chenhansunding@gmail.com**



Demo



Thank You!