OOC

A wonderful adventure in high-level languages land

Avant-propos and kickstart

This is the first paper on the ooc language, its first implementation, and the story and motivation behind it. It is intended to give an overview of the history of the project, and hints on plans for the future.

While this paper was primarily written for a french-speaking school work, it is written in english, for the simple reason that it appeals to a greater audience.

It is a well known and understood fact that programmers and, in a broader sense, computer scientists, have difficulties or are at least reluctant to learn about new languages different from what they work with everyday.

ooc is not about providing a silver bullet. It's about trying to take a step in the right direction. It is with an open mind that one should read this document, or not at all. (Keep trolls outside) *If you think it's stupid, maybe it's just not for you.*

I. The history

Grew out of the frustration to code a school projet in C, missed OOP, wanted the project to be fun, not boring.

First implementation became reasonably usable in two months, then added features, bug fixes, documentation, etc.

Now open source, source available @googlecode, contributions very welcome!

II. A Tale of Two Mindsets

1) Optimization vs Portability (C/C++ vs Java)

– Even today, most C-inclined programmers don't distinguish between Microsoft's Java 1.1 naive implementation, and Sun's state-of-the-art JIT-enabled Hotspot 1.6 Server JVM.

– « C is portable » is a joke. SDL/sdl_stdinc.h Architecture-specific, OS-specific, Compiler-specific. « C has been ported » is more correct. The 64-bit shake-up.

2) Don't be so static (or why Python is the new bash)

– Real-world case study: Pidgin crashes randomly, Emesene displays stack traces and graciously reloads. *When performance is not an issue, high-level is a significant advantage.*

– Lesson to be learned: build an inverse pyramid. Small, rock-solid core, Medium-sized reliable APIs, Big, real-world, « coded by humans » applications. *It's good to know you can rely on the ground to keep you from falling.*

3) Modularize or die

- Java is bloated: Linux JDK = 76.42 MB, Linux JRE = 19.24 MB (source: java.sun.com, 2009-05-18). *Java is not a language, it's a platform.* J2SE/J2EE/J2ME mess.
- Size & Complexity are obstacles to portability: The FreeBSD hell (and why the demon mascot is to the point) 300Mb+ downloads, 2.5 Gigs for compilation
- Module distribution: Java=FAIL (monolithic), Perl shows the way (CPAN), Ruby is a good student (Gems), etc. JRE 6 Update improvements: *modularity must be in the design, from the start.*

III. The design decisions trial

Programmers always have to justify for their choices. Better anticipate.

1) Brace yourself!

- Ever opened a python file with mixed space/tabs? Ever added an instruction to a one-line if without braces?

2) Prototype early, class often.

- Static checking goodness, if it walks like a duck and talk like a duck, we cross our fingers and pray that it's a duck. Why JavaScript increases web developers' death rate due to stress.
- **The Nice Programming Language,** http://nice.sourceforge.net/ shows the way: allow to add methods to existing classes, maintain safety.

3) Simple vs Verbose, Short vs Readable

- Java's repulsion towards nice shortcuts. Why no closures ? :(Lisp-ers are laughing. Excellent IDEs are an excuse to bad language design.
- KISS: Keep It Simple, Stupid!
  - #include <string.h> instead of <string>, and get 30 lines of unreadable template error messages for free. http://www.gamedev.net/community/forums/topic.asp?topic_id=292014
- DRY: Don't Repeat Yourself.
  - ```
    class AnyClass {
          private final static Logger logger;
    ```
    ```

```
        static {
            Logger.getLogger(AnyClass.class);
        }
    }
```

- Default initialization/copy constructors in C++ are evil
- foreach is good.

4) C++ semantics are clearer than its syntax

- « pure virtual » = 0, probably due to Bjarne Stroustrup's math degree ;)
- C++0x will raise the bar for obfuscation contests. Straight from B. Stroustrup's page http://www.research.att.com/~bs/C++0xFAQ.html
  - `int x1 = {7.3}; // not an array literal: prevents narrowing`
  - `unsigned char c [[ align(double) ]] [sizeof(double)];  // array of characters, suitably aligned for a double (looks like bash to me)`
  - `struct foo* f [[carries_dependency]] (int i); // hint to optimizer`
- See the bright side too: stole `auto` from D, in-class initializers from Java.
- Conclusion? Steal from Java, squeeze out the nonsense, pour syntactic sugar, and serve it cold.

5) Coding the first implementation in Java

- How can you use something you criticize so much? Love/hate relationship.
- Somewhat portable, high-level enough, good collections & I/O APIs, what more can you ask for?
- Eclipse is *great* for refactoring, helps cleaning the 4AM messes.
- Had to start from something, just felt more comfortable in that language.

6) Using C/GCC as a backend

- Has been ported to 50+ architectures, including ARM, Atmel AVR (embedded), IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PowerPC, SPARC, VAX, even on my TI-89!
- Open-source, free, good general optimizations, mature, etc. Mainstream, people are used to it. *If it builds all my gentoo ebuilds, it's good enough for ooc.*
- Still open (additions encouraged!) MSVC/Watcom C/TinyCC support, etc. Did consider C--, but not mainstream/maintained enough.

7) Garbage Collection

– Boehm GC rocks in the role of the state-of-the-art conservative garbage collector. Small overhead, big relief. Future: precise garbage collection? Needs more integration effort!

– Crash course in Garbage Collectors for C/C++ zealots: http://icu-project.org/docs/papers/cpp_report/an_introduction_to_garbage_collection_part_i.html

8) No VM?

– Don't have the time/the knowledge to write a good one, pretty much defeats the « be small » goal, overkill for ooc's initial goal.

– Not excluded from the future

IV. The State Of The Llama

1) A few stats

– Size: roughly 16K lines according to wc, 10K lines according to ohloh.

– $115K development cost, according to ohloh (note: young projects are always over-evaluated) http://ohloh.net/p/ooc-language

2) Features

– Packages, source units

– Method polymorphism (same name, different arguments), correct name mangling and compile-time dispatch

– Abstract classes, simple inheritance

– Foreach, ranges

– Null checks, type safety

– Library management (glut example)

– GCC and Make backends

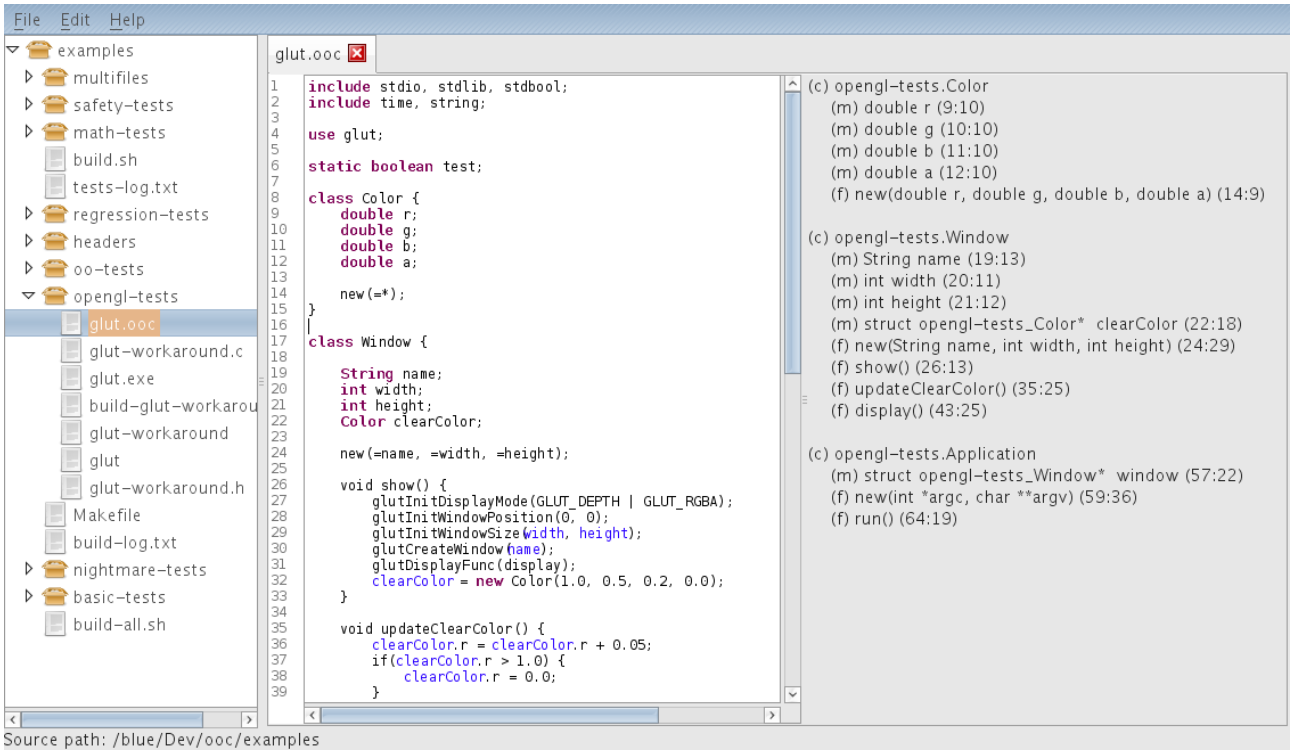
3) SDK

– I/O media-independant writing (e.g. file, memory, network connection)

– Collections: ArrayList, SparseList

4) IDE support

Java/Swing IDE project: eon



VIM syntax highlighting

```
include stdio, stdlib;

class Class {
        int a;
        double b;
        char c;

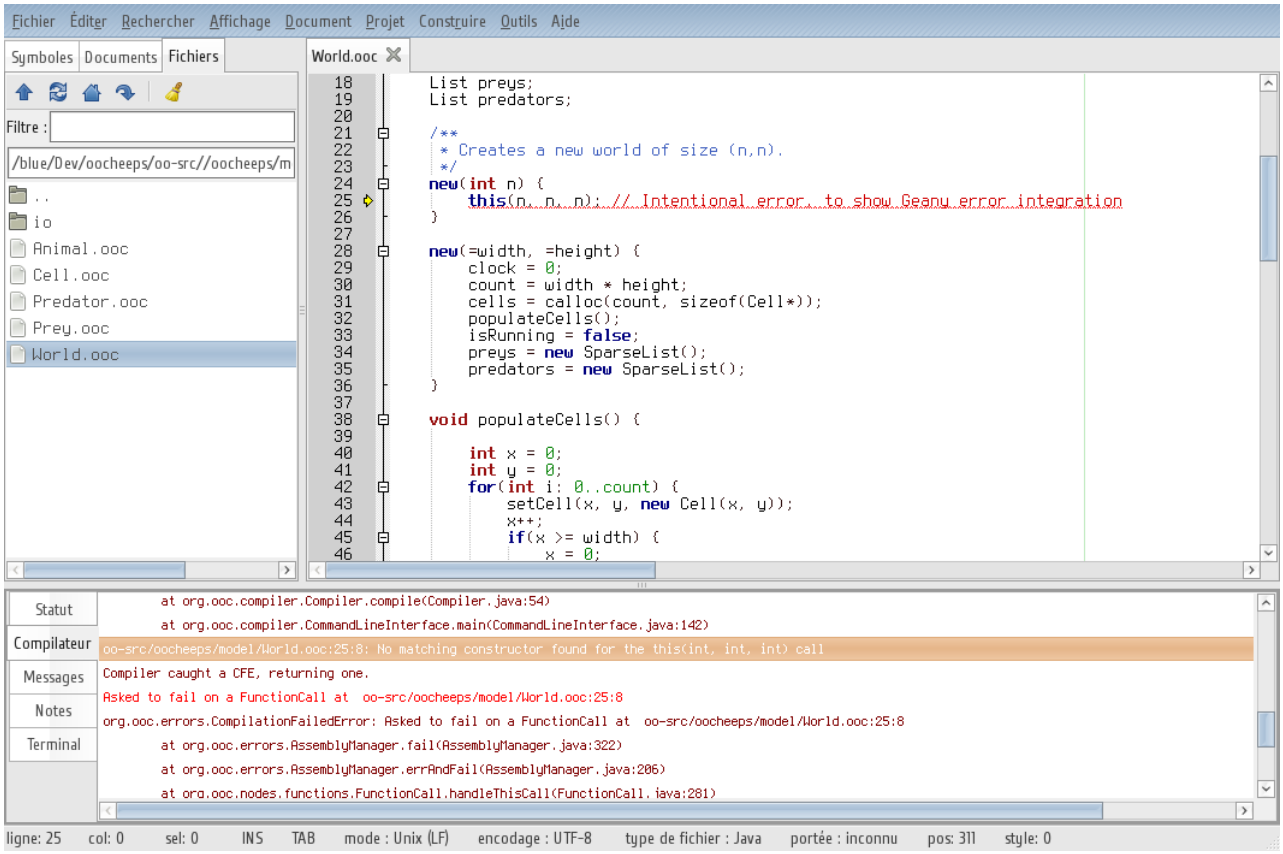
        new(=*);

        new() {
                this(12, 32.4, 'f');
        }

        void print() {
                printf("%d, %f, %c\n", a, b, c);
        }
}

int main() {
        new Class().print();
}
~
~
~
~
-- INSERTION --                                  4,2-9         Tout
```

Geany integration



5) Portability

– Tested under:
  – Gentoo / Intel 32bits (Core 2 Duo)
  – Windows XP|MinGW / Intel 32bits (Core 2 Duo)
  – RedHat / Sun Server (COSUN rooms at the EPFL)
  – FreeBSD: abandoned at the JDK installation
  – OpenSolaris: still some work for libraries detection

V. Short ooc tutorial

Blah blah blah

VI. Under the hood: the nasty stuff

Blah blah how we abuse structure for inheritance, etc.

VII. Looking to the future

Blahblahblah, how we hope someone might one day get truly interested, and contribute, blahblah and how I'm to quickly become rich by writing unacceptably long books and giving conferences and seminaries