THE SATURDAY EVENING POST
They call him GINGER
HE'S GOT SO MUCH OF IT!
"CANADA DRY" PALE GINGER ALE
Ginger means zest. That's why we say, for a quick pick-up ... or bracer ... an ice-cold refreshment, call for a tall, cool glass of Canada Dry! Its famous secret process brings out the full, rich flavor of the world's best ginger root! With meals ... in mid-afternoon ... whenever you want a lift, drink Canada Dry! It's gingervating!
Drink Canada Dry Ginger Ale
IT'S GINGERVATING!
Picks you up . . . aids digestion

!

# Javolution C++

## *They call him Ginger!*

***« It looks like Java, it tastes likes Java… but it is C++ »***

**October 20, 2012**

# What is the problem?

- More and more hybrid C++/Java projects
  - Developer expertise required in both Java and C++
- C++ total cost is significantly higher
  - But cost of migrating existing C++ components to Java is prohibitive.
- Standardized and well established software practices exist in the Java world
  - C++ developers are on their own (multiple solutions to address the same problems lead to additional complexity)
- Many Open-Source implementations of Software Standards exist only in Java
  - OSGi, GeoAPI, UnitsOfMeasure, etc.

# Many causes of variability.

- Developers expertise varies considerably.
- Testing performed at the end (integration) due to component inter-dependencies.
- Insufficient documentation.
- “Not Invented Here” Syndrome.
- Proprietary solutions not maintained which later become legacy burden.
- It is very beneficial to follow well-established standard specification.

***“Doing the right thing is difficult, but doing it right is easier.”***

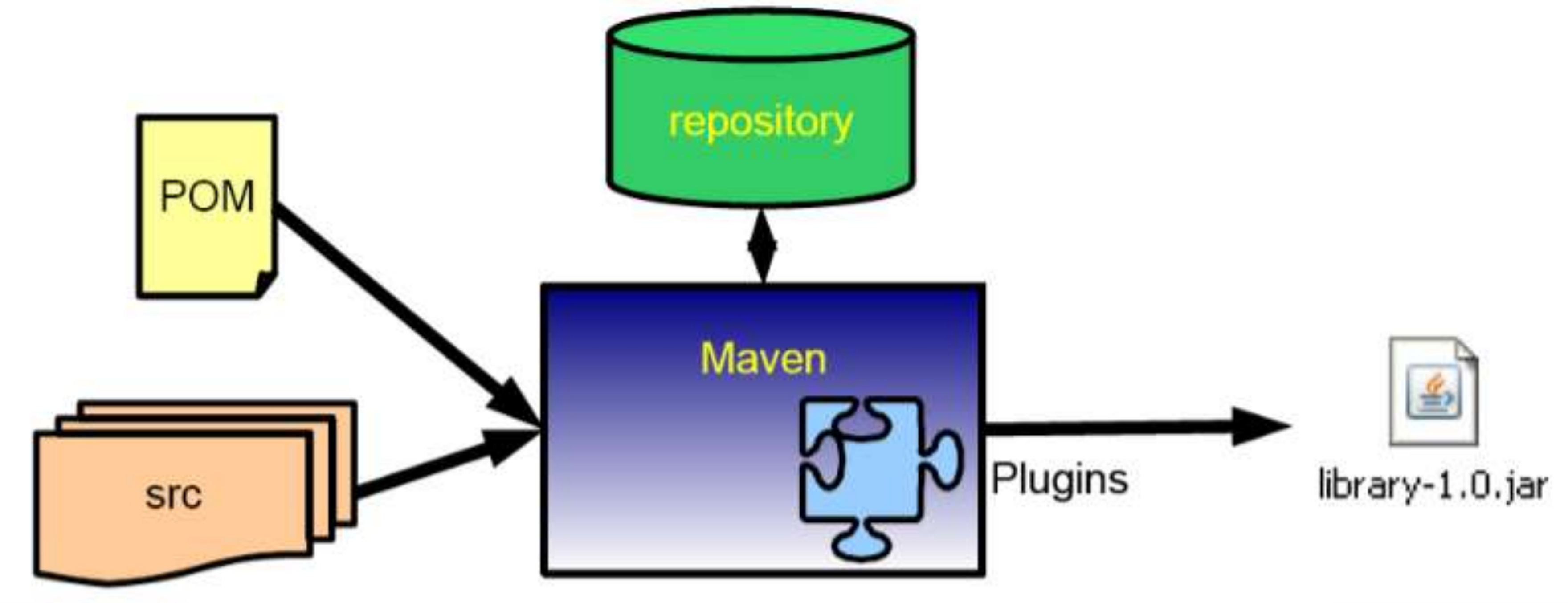
# Javo(So)lution.

- Uniformization of C++/Java development through the use of a common framework (Javolution) based on Java standard library specification.
- Facilitating the migration of Java OSS code to C++
- Promote the “Service Oriented Approach” by providing an OSGi framework for both Java and C++
- Reduce documentation by having the same specification/design for our Java and C++ components.
- Unification of code building for Java and C++ (maven used for both).

# Maven Build

- **Apache Maven** (maven native plugin) is used to produce artifacts (dynamic libraries, static libraries, executable) and to perform unit tests.

- Profiles and packaging classifiers are used to address platform variability (windows, linux, etc.)



# What is Javolution C++ ?

- A mirrored C++ library sharing the same specifications, documentation and unit testing as its Java pendant.

- A “behind-the-scenes” C++ infrastructure based on smart pointers (real-time garbage collection through reference counting).

- Integrated memory cache making small, short lived objects (e.g. value types) very efficient.

- C++ packages/classes derived from standard Java (e.g. javolution::lang, javolution::util)

- A C++ dynamic execution and testing framework (OSGi & JUnit) identical to Java.

# C++ Class Definition

## The general pattern for class/interface is as follow:

```
#include "javolution/lang/Object.hpp"

namespace com {  namespace bar {
    class Foo_API; // Value type (used to define the API)
    typedef Type::Handle<Foo_API> Foo; // Reference (same as Java)
}}

class com::bar::Foo_API : public virtual javolution::lang::Object_API {
private:
    Param param;

protected:
    Foo_API(Param const& param) { // const& for handles parameters.
        this->param = param;
    }

public:
    static Foo newInstance(Param const& param) { // Generalized use of
        return new Foo_API(param);               // factory methods.
    }
    virtual void fooMethod () { ... };

}
```

# C++ Parameterization – Better than Java!

- Unlike Java, C++ class parameterization is not syntactic sugar but efficient use of C++ templates!

- All javolution::util collections are parameterized.

```
List<String> list = FastTable_API<String>::newInstance();
list->add(L"First");
list->add(Type::Null);
list->add(L"Second");
```

- Also used for Java-Like Enums

# Synchronization

- Supported through a macro: synchronized(Object) mimicking the Java synchronized keyword.

- Can be performed on instances of Javolution collections and Class (for static synchronization).

```
synchronized (trackedServices) {// trackedServices instance of FastMap
   for (int i = 0; i < serviceReferences.length; i++) {
       Object service
           = actualCustomizer->addingService(serviceReferences[i]);
       trackedServices->put(serviceReferences[i], service);
   }
   trackingCount = 0;
}
```

# Miscellaneous

- **Limited reflection support through RTT**
- **Auto-boxing of primitive types (boolean, integer, float, wide strings).**

```
Integer32 i = 23;
Float64 f = 3.56;
Boolean b = true;
String s = L"xx";
```

- **All variables are initialized to Type::Null (NullPointerException if not set before use).**
- **Wide-String (literal) concatenation supported.**

```
throw RuntimeException_API::newInstance(
    L"Bundle " + symbolicName + L" not in a resolved state‶ );
```

- **Dynamic length array Type::Array<type>**

```
Type::Array<ServiceReference> serviceReferences
    = context->getServiceReferences(serviceName, Type::Null);
if (serviceReferences.length == 0) return;
```

# Minor differences with Java

- No ‘finally’ keyword in C++ (but try…catch same as Java).
- Static methods are called using the name of the class with the suffix ‘_API’
- Generalized use of static factory methods, e.g. MyClass_API::newInstance(…)
- Synchronization not supported on every object but only on those whose class implements the Object_API::getMutex() virtual method.

# What next?

- Automatic translator (JavaCC based) of Java source code to Javolution C++

- More Java library conversion (e.g. OpenSDK, JScience, …)

- Help wanted in writing the translator tool ☺