Introduction to Gosu

A New Language for the JVM
from Guidewire

Scott McKinney
Guidewire Software
smckinney@guidewire.com

www.**gosu-lang**.com

Today's Discussion

Gosu

- Background
- Highlights
- Open Type System
- Language Features
- Tools
- Q&A

Background

- **Guidewire**
  - Provide large scale, highly configurable applications
  - Gosu enables unified configuration of customer facing resources:
    - Rules, Workflows, Web pages, Messaging, Web-services, Tests, etc.
- **Language History** (2002 – present)
  - No statically typed, embeddable scripting language available
  - Started small as a rule expression language
  - Evolved slowly: Scripting → OOP → Open Type Sys → Bytecode
- **Roots**
  - Influenced by Java, C#, EcmaScript, Ruby, Pascal
  - Static type system an absolute requirement, esp. for *tooling*
  - Ideals: Pragmatic, Versatile, Professional

Highlights

Gosu

- **Open Type System** (type ***plug-ins***)
- **Object Oriented** (**mainstream**)
- **Imperative** (***familiar*** grammar)
- **Statically Typed** (type-safety, ***tooling!!!***)
- **Type Inference** (**concise, readable,** dynamic feel)
- **Reified Generics** (plus **simpler** array-style variance)
- **Enhancements** (add behavior to existing types)
- **Closures**/Blocks (**productive** with collections)
- **Composition**/Delegation (**mix-in** support)
- Lots (and *lots*) of useful **language features**
- **Eclipse** Plug-in (***full*-featured**)
- **Interactive Command Line** (Read Eval Print Loop)
- ***Fast*** – (On par with Java, conventional bytecode)
- **Embeddable** and Standalone

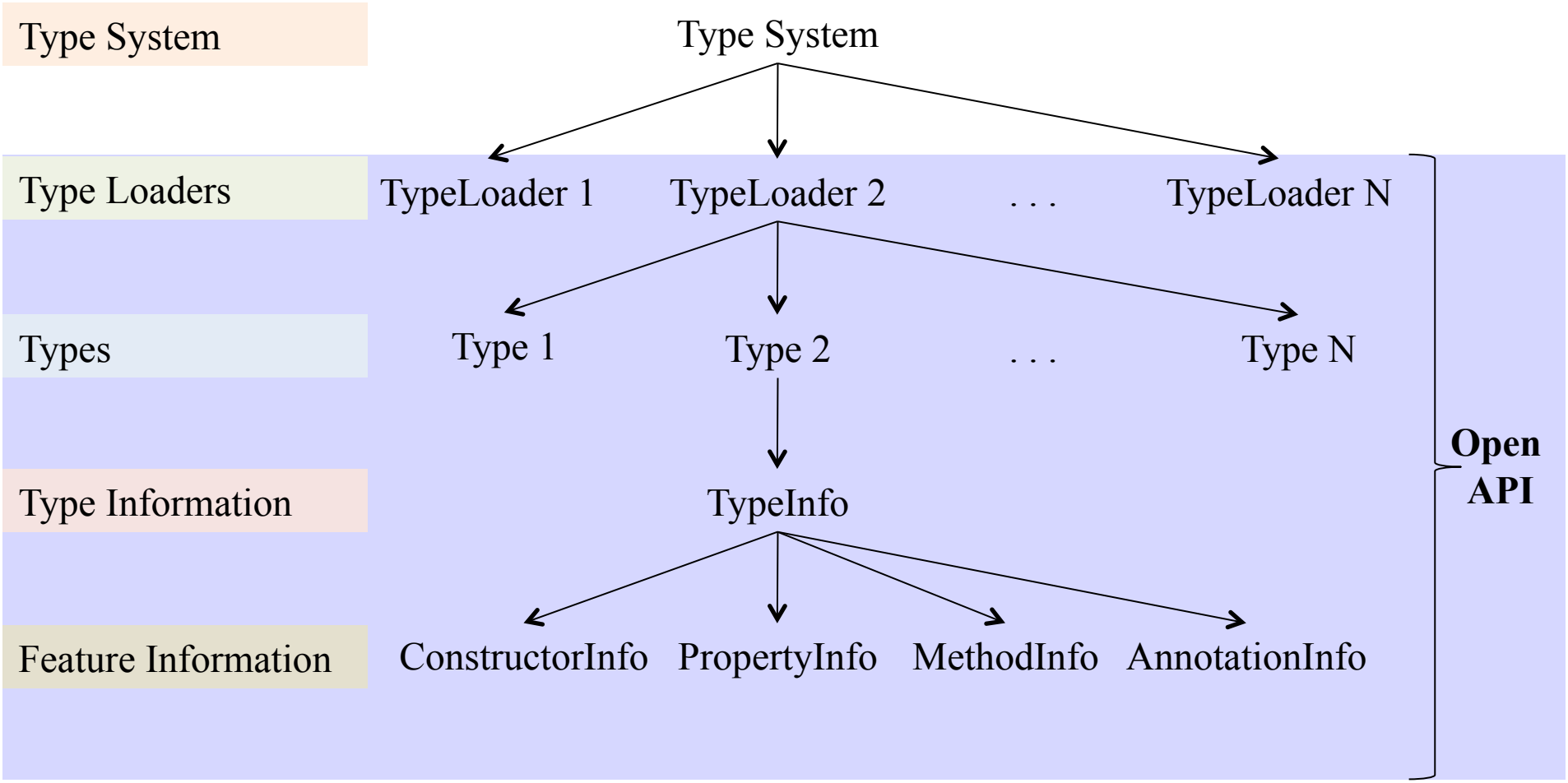
Open Type System

- Open Type System = *Pluggable* Type System
- Open API for defining custom types
- Abstractions for type loading and type information
- Avoids messy code generation
- Alternative to DSL → Domain Specific ***Type*** (DST)
- Examples of Types:
  - Gosu & Java
  - Templates
  - XSD / XML
  - Web Services / WSDL
  - Database Entities
  - Web Pages
  - Name Resources
  - Etc.

Included in open source

Internal to Guidewire

Type System Structure



Open API

Gosu

ITypeLoader

```
IType getType( String name )
. . .
```

IType

```
ITypeInfo getTypeInfo()
. . .
```

ITypeInfo

```
List<IPropertyInfo> getProperties()
List<IMethodInfo> getMethods()
List<IConstructorInfo> getConstructors()
. . .
```

IMethodInfo

```
IMethodCallHandler getCallHandler()
. . .
```

IMethodCallHandler

```
Object handleCall( Object ctx, Object… args )
. . .
```

Included Types

Gosu provides several kinds of types in the current open source release. These include:

- Classes, Interfaces, Enums
- Enhancements
- Programs
- Templates
- Blocks/Closures
- XSD / XML
- Web Services / WSDL
- Dynamic Type*

Language Features

Obligatory "Hello World"

Brace for it…

Hello World

Hello.gsp

```
print( "Hello World!" )
```

```
> gosu Hello.gsp
Hello World!
>
```

Gosu runs ***Programs***

- No boilerplate class
- No main() method
- More on programs later…

More on Command Line Gosu later…

Back to Langauge Features…

Object Oriented

Gosu

- **Superset** of Java's OO capabilities
- Fully **compatible** w/ Java
- Single class, multiple interface **inheritiance**
- **Composition**/Delegation support
- **Properties**
- **Annotations**
- **Enhancements** (add behavior to existing types)
- Anonymous types with **variable capture**

Class Structure

Gosu

```
package demo

uses java.util.List
...

class Foo
{
  var _name : String
  construct() ...
  function foo() ...
  property get Name()
  class Inner ...
}
```

- `package` keyword same as Java
- `uses` keyword = Java import
- `class` public by default
  - fields
  - constructors
  - methods
  - properties
  - inner types

Programs

Gosu

- Gosu **executes** Programs/Scripts
- **No more boilerplate** class with static main()
- Can be a simple **expression** or…
- Can define any number of **statements**
- Can define **functions and properties**
- Can define **classes**, inner classes, and **closures**
- Type-safe access to **command line** arguments

Programs

Gosu

```
uses javax.swing.JFrame
uses java.awt.Rectangle

var frame = new MyFrame()
showMyFrame()

function showMyFrame() {
  frame.Visible = true
}

class MyFrame extends JFrame {
  construct()  {
    super( "Hello" )
    DefaultCloseOperation = DISPOSE_ON_CLOSE
    Bounds = new Rectangle( 100, 100, 100, 100 )
  }
}
```

- Mix **statements** with…
- **Functions** and…
- **Classes**

Templates

Gosu

- Are types e.g., `com.foo.SomeTemplate`
- Support JSP/ASP-style syntax: `<%, <%=, etc`
- Support `${` syntax too
- Declare parameters: `<%@ params( x: String ) %>`
- Can be used anywhere
- Are supported in String Literals

Templates

MyTemplate.gst

```
<% uses java.util.Date %>
<%@ params(x : String) %>

Template text

<% if( x.HasContent ) { %>
Invisible template text
<% } %>

<%-- comment --%>

The param is: ${x}
```

Usage:

```
// Render directly to String
var str =
 MyTemplate.renderToString( "hello" )

// Render to writer
MyTemplate.render( _writer, "hello" )

// Embedded in a String literal
var str =
  "The date is ${new Date()}"
```

Generics

- ***Reified!***

```
var foo = new Foo<String>
print( foo typeis Foo<String> ) // true!
```

- Array-style Variance (no wildcards, easy to *understand*!)

```
var l : List<CharSequence>
l = new ArrayList<String>()
```

- Type parameters of *non-bytecode* types

```
var l = new ArrayList<xsd.abc.Person>()
```

- Fully Compatible with Java Generics

```
class Album<T extends Photo> extends ArrayList<T>
```

Enhancements

Gosu

```
package demo

enhancement MyListEnhancement<T> : List<T> {
  ...
  function each( visit(item: T) ) { ... }
}

e.g.,
var list = {"Pascal", "Java", "Gosu"}
list.each( \ e -> print( e ) )
```

- **Add methods** and properties to existing types (including Java)
- Look and **feel like a class**, implicit 'this' reference
- Fully support **generics**
- Don't modify existing class; **just type information**
- Statically dispatched
- Can't add state – **no fields** allowed

Closures/Blocks

Let's turn a list of Strings into a list of their lengths . . .

```
var list = {"Pascal", "Java", "Gosu"}

// Collections with blocks (and enhancements)...
var list2 = list.map( \ e -> e.length() )

 // The declaration site looks like:
 function map<Q>( mapper(elt : T):Q ) : List<Q> { . . . }
```

- Anonymous functions declared inline
- Can be either expressions or statement lists
- Argument and return types **inferred** based on context
- True closures with **captured variables**
- Invoked like normal functions
- **Covariance** on return types, **contravariance** on argument types

Type Inference


```
var list = {"Pascal", "Java", "Gosu"}
```

Life *without* type inference…

```
var list2 : List<Integer> =
  list.map<Integer>( \ e : String -> e.length() )
```

- The String type is inferred from the map() method
- map()'s type parameter is inferred from the block's return type
- Finally, list2's type is inferred from map()'s generic return type

```
var list2 = list.map( \ e -> e.length() )
```

Composition/Delegation

Gosu

Composition in **Gosu…**

```
// Sample Mixin interface
public interface IClipboardPart
{
  boolean canCopy();
  void copy();
  boolean canPaste();
  void paste();
  . . .
}
```

```
class MyWindow extends Window implements IClipboardPart
{
  delegate _clipboardPart represents IClipboardPart
  ...
}
```

- `delegate` keyword
- `represents` clause specifies interfaces, owning class must declare
- Compiler automatically dispatches to delegate's implementation
- Otherwise behaves just like a field
- A single delegate can represent multiple interfaces

More Language Features

Gosu

- Context sensitive **eval()** support (in a static language? really!)
- Using-statement – supports **RAII** (Resource Acquisition Is Initialization)
- Map and **Collection initialization** syntax
- **Object initializer** syntax
- Extensive **interval support** e.g., 1..10
- **Enhanced switch**-statement (any type, any expression)
- **Smarter for-statement** (handle more types, provides index)
- **Associative array** syntax for dynamic access to properties
- **Null short-circuit** in property access expressions
- **No checked exceptions!**
- **Etc.**

Tools!!!

Gosu

In our view:

- A professional, general purpose language is impractical without full-featured **IDE support**
- The larger the project, the greater the **pressure on tools**
- Huge productivity gains via build-time **type-safety**, editor **feedback**, **code completion**, **navigation**, **usage searching**, and **refactoring** are not achievable without the ability to perform **solid static analysis**

Good news:

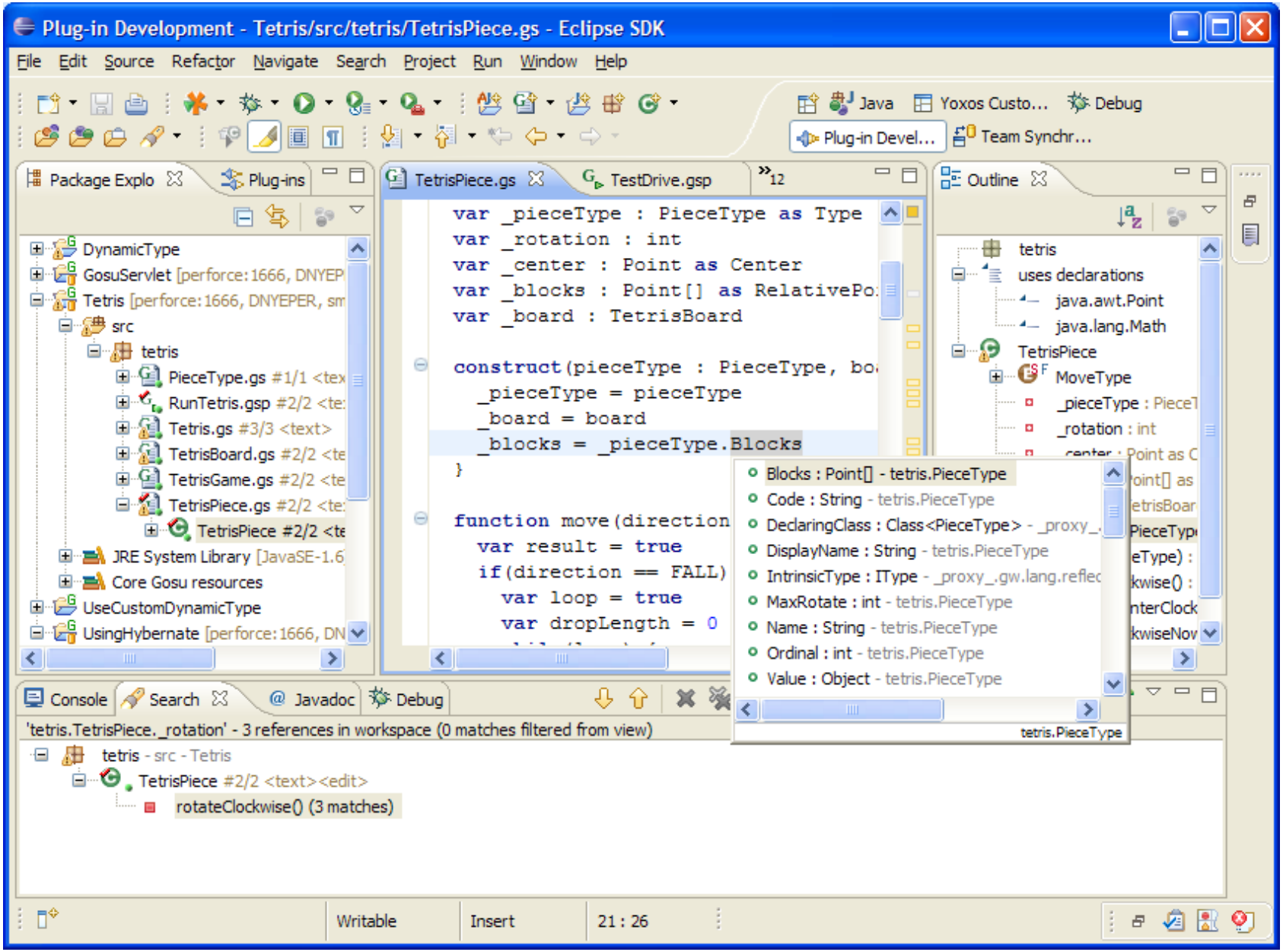
- Gosu's type system enables a rich set of static analysis tools
- We've been busy…

Eclipse Plug-in

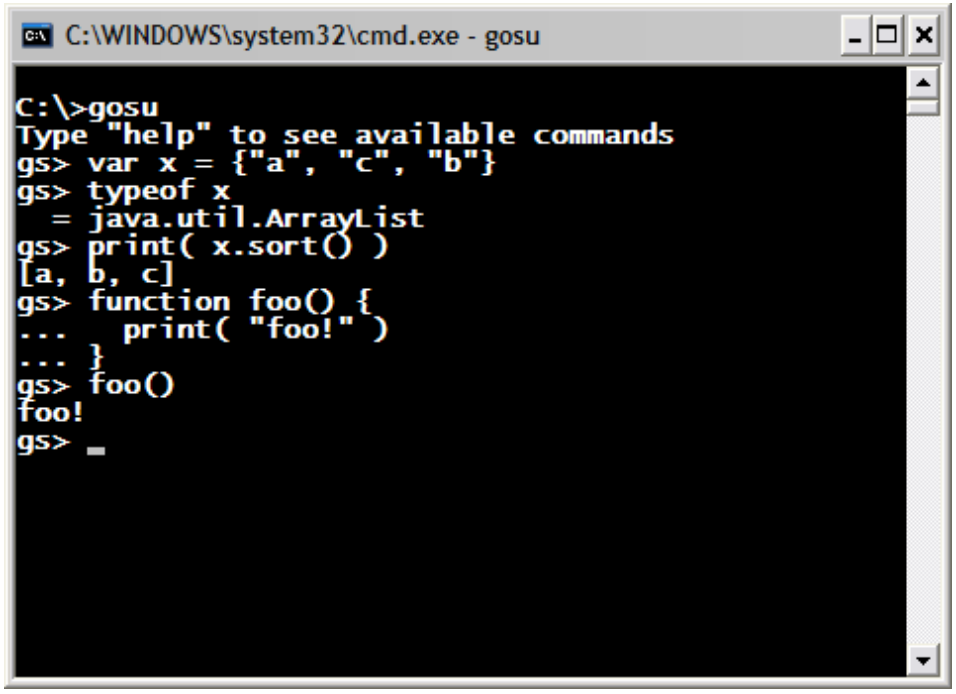
Full-Featured

- Syntax coloring
- Instant feedback as you type
- Code completion
- Code navigation
- Member Usage Search
- Type Usage Search
- Refactoring
- Hover text
- Structure views
- Occurrence highlighting
- Full featured debugger

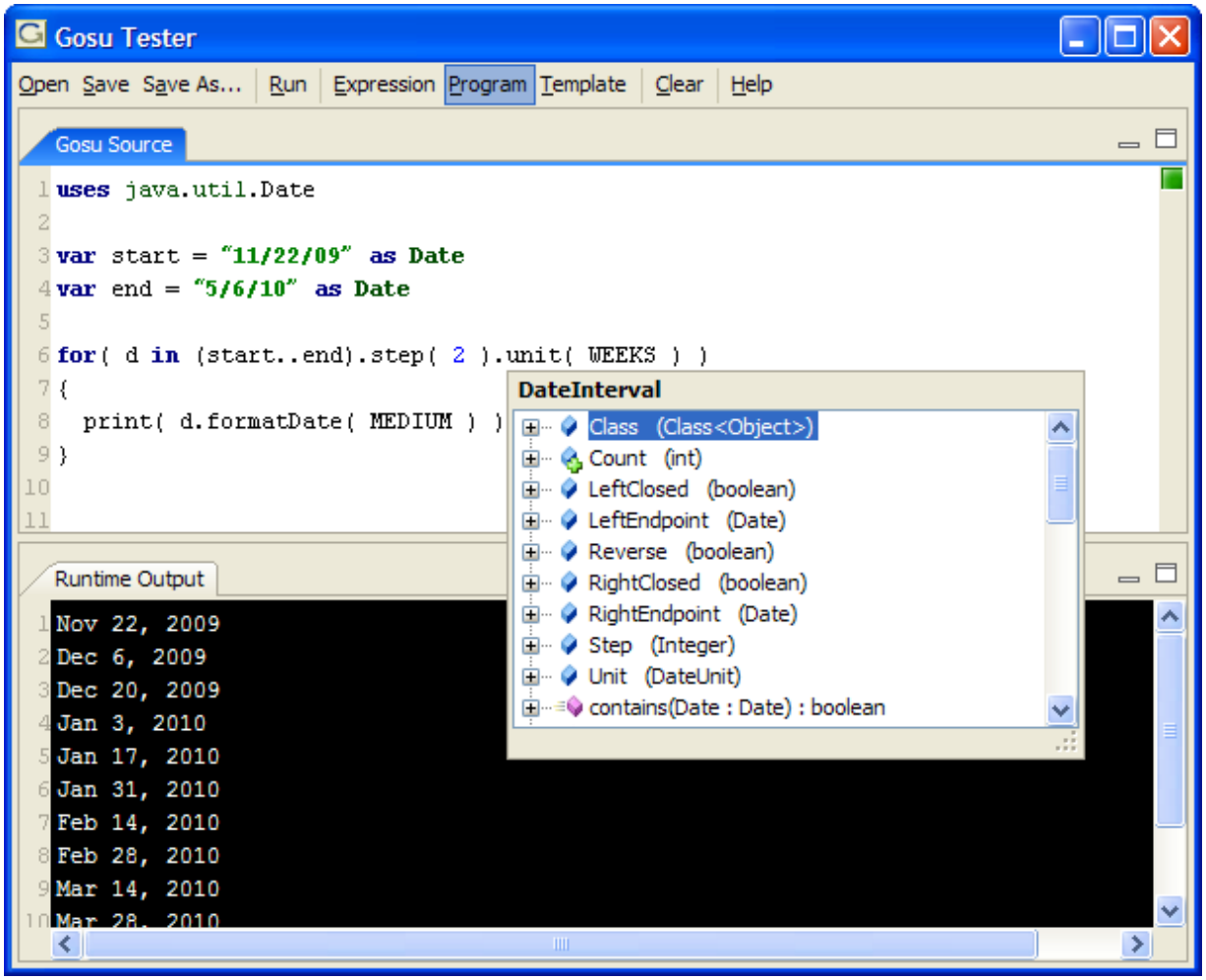
Eclipse Plug-in



REPL Command Line



Interactive Script Editor



Q&A