

## Wollok: en el aula y más allá

Javier Fernandes<sup>1,2</sup>    Nicolás Passerini<sup>1,2,4</sup>

Pablo Tesone<sup>3,1,2,4</sup>    Débora Fortini<sup>1,4</sup>

Nahuel Palumbo<sup>4</sup>    Juan Contardo<sup>4</sup>    Carlos Raffellini<sup>4</sup>

<sup>1</sup>Universidad Nacional de Quilmes

<sup>2</sup>Universidad Nacional de San Martín

<sup>3</sup>Universidad Nacional del Oeste

<sup>4</sup>Universidad Tecnológica Nacional - F.R. Buenos Aires.

Workshop de Ingeniería en Sistemas y Tecnologías de la Información  
19/09/2015

# Agenda

- 1 Introduccion
- 2 Desarrollo de Wollok
- 3 Features Avanzados
- 4 Wollok Game
- 5 Experiencia en el Aula

# ¿Por qué es difícil aprender OOP?

- Enfoque en un lenguaje particular
- Demasiados conceptos

```
package examples;  
  
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

- Entornos de desarrollo limitados o inadecuados
- Aprender a programar exige **aprender a abstraer**

## ¿Por qué es difícil aprender OOP?

- Enfoque en un lenguaje particular
- Demasiados conceptos

```
package examples;  
  
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

- **Entornos de desarrollo** limitados o inadecuados
- Aprender a programar exige **aprender a abstraer**

# Antecedentes

## Ozono

<http://ozono.uqbar-project.org/>

- Basado en Smalltalk
- **Recorrido incremental**
  - Metamodelo simplificado: sin clases ni herencia
  - Foco en objeto - mensaje - referencia - polimorfismo
- Herramientas de visualización de código
  - Diagramas de objetos / clases

## Gobstones

<http://www.gobstones.org/>

- Cuidadosa selección de los elementos sintácticos
- Elimina la necesidad de entrada-salida
- Separación entre elementos con efecto y elementos puros

# Antecedentes

## Ozono

<http://ozono.uqbar-project.org/>

- Basado en Smalltalk
- **Recorrido incremental**
  - Metamodelo simplificado: sin clases ni herencia
  - Foco en objeto - mensaje - referencia - polimorfismo
- Herramientas de visualización de código
  - Diagramas de objetos / clases

## Gobstones

<http://www.gobstones.org/>

- Cuidadosa selección de los elementos sintácticos
- Elimina la necesidad de entrada-salida
- Separación entre elementos con efecto y elementos puros

# ¿Qué es Wollok?

- Lenguaje + muchas herramientas
- Optimizados para la enseñanza
- Cercanos a las herramientas profesionales *mainstream*
- Proyecto abierto

## ¿Qué es Wollok? - Pensado para enseñar

- Sintaxis cuidada
  - selección de keywords
  - return obligatorio
  - énfasis en objetos y mensajes (aunque no todo es objeto mensaje)
- Combina object-based con class-based programming
- APIs minimalistas (ej: colecciones)
- Import system



## ¿Qué es Wollok? - Cercano al mainstream

- Ambiente de objetos basado en archivos
- Framework de testing integrado
- Literales para listas y conjuntos (próximamente diccionarios)
- Manejo de excepciones

# Desarrollo de Wollok

- OpenSource: LGPLv3
- Stack: Eclipse XText + Xtend Lang
- SCM:
  - **Código:** GitHub ([uqbar-project/wollok](https://github.com/uqbar-project/wollok))
  - **Build:** Maven + Tycho
  - **Continuous Integration:** Travis
  - **Continuous Deployment**
  - **Coverage:** coveralls + jacoco
- Testing & TDD

# Desarrollo de Wollok

## Continuous Integration & Deployment

- **GitFlow**
  - Feature Branches
  - Pull-Requests
  - *dev* → *master* ← *hotfixes*
- **Integration:**
  - Travis
  - compile, test, coverage, deploy
- **Deployment:**
  - **Productos** (IDE): multiples plataformas
  - **Update Sites**
  - **WDK**
  - 2 Ambientes: Stable & Dev

# Desarrollo de Wollok

## Testing & TDD

- 87% Cobertura
- **Runtime**
  - Testean ejecución
  - Interprete
  - **JUnit + iDSL**
- **Estáticos**
  - **Chequeos:** XPect
  - **Type System:** JUnit + iDSL
  - **Autocomplete:** XPect
  - **Formateo:** JUnit + iDSL
- **Pendientes**
  - Quick-Fixes
  - Refactors

# Testing & TDD

## Runtime

Testeo del intérprete

```
class PostFixOperationTestCase extends AbstractWollok
```

```
@Test
```

```
def void testPlusPlus() {'''
```

```
  program p {
```

```
    var n = 1
```

```
    n++
```

```
    assert.that(n == 2)
```

```
  }'''.interpretPropagatingErrors
```

```
}
```

## Testing & TDD

### Cheques Estáticos

Testeo de Cheques Estáticos

```
/* XPECT_SETUP org.uqbar.project.wollok.test
```

```
class Golondrina {  
    var energia = 100
```

```
    method energia() {
```

```
        // XPECT errors --> "Cannot assign a  
        energia = energia
```

```
}
```

## Testing & TDD

### Autocomplete

#### Testeo del Autocomplete

```
class Golondrina {  
    var energia = 100  
    method getEnergia() {  
        /* XPECT proposals at 'energia' ---  
           "Value", #, (  
           Create Object - Create a new object  
        --- */  
        return energia  
    }  
}
```

# Testing & TDD

## Formatter

### Testeo del Formatter

@Test

```
def void testSimpleProgramWithVariablesAndMessages  
    assertFormatting("""program p { val a = 10 val  
        ""
```

```
        program p {  
            val a = 10  
            val b = 20  
            this.println(a + b)  
        }""")
```

```
}
```



# Testing & TDD

## Type System

### Testeo del Type System

@Test

```
def void testNumberLiteral() { '''program p {  
    val a = 46  
    val b = "Hello"  
    val c = true  
}'''  
    .parseAndInfer.asserting [  
    assertTypeOf(WInt, "val a = 46")  
    assertTypeOf(WString, 'val b = "Hello"')  
    assertTypeOf(WBoolean, "val c = true")  
    ]  
}
```

# Features Avanzados

- Debugger
- Tests
- Sublime Plugins
- I18N

# Debugger

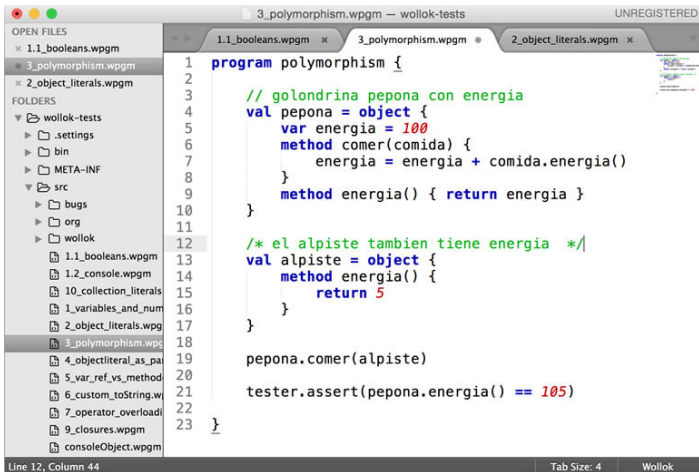
- UI integrada a Eclipse Debug
- Breakpoints: agregar, remover, deshabilitar, etc
- Step, into, out
- Inspeccionar variables

# Sublime Support

- WDK
  - No IDE
  - ~ 70MB (vs ~ 140)
  - Headless: wchecker, winterpreter, wtest
- Syntax highlight
- Templates
- Linter

# Sublime Support

## Syntax Highlight

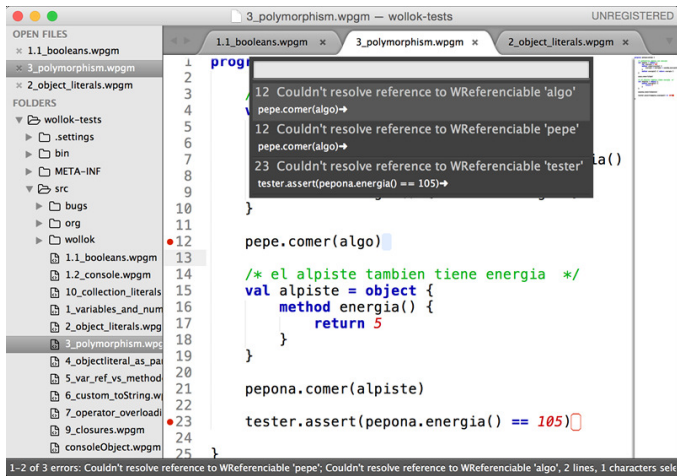


The screenshot shows the Sublime Text editor interface with a file named `3_polymorphism.wpgm` open. The left sidebar displays the 'OPEN FILES' and 'FOLDERS' panels. The main editor area shows Wollok code with syntax highlighting: keywords in blue, comments in green, and literals in red. The code defines a `polymorphism` program with two objects, `pepona` and `alpiste`, each having an `energia` attribute and a `comer` method. The `comer` method for `pepona` increments the energy of the food it consumes. The `tester` object asserts that `pepona.energia()` is equal to 105 after consuming `alpiste`.

```
1 program polymorphism {
2
3     // golondrina pepona con energia
4     val pepona = object {
5         var energia = 100
6         method comer(comida) {
7             energia = energia + comida.energia()
8         }
9         method energia() { return energia }
10    }
11
12    /* el alpiste tambien tiene energia */
13    val alpiste = object {
14        method energia() {
15            return 5
16        }
17    }
18
19    pepona.comer(alpiste)
20
21    tester.assert(pepona.energia() == 105)
22
23 }
```

Line 12, Column 44

# Sublime Support Linter



# Wollok Game

- s
- a
- b
- o

# Experiencia en el Aula

- S
- a
- S



# Muchas gracias