Meta-*, Reflexión y Anotaciones

Mecanismos para generación de frameworks de aplicaciones empresariales

Agenda

- Discusión sobre Meta-*
- Reflexión
- Anotaciones
- POJOs
- Inversión de Control

Meta - *

Definición de Meta

- Meta: del griego antiguo μετά (meta).
    - Trascendente, o que abarca
    - Perteneciente a un nivel arriba o más allá
- Ejemplos en ciencias de la computación:
    - Meta-data
    - Meta-Lenguaje
    - Hilbert, Gödel, Turing
    - Teoría del constructor

Meta Data

- Datos acerca de los datos
- Ejemplos
  - Fecha de creación, tamaño, fecha de modificación de un archivo en un sistema de archivo del OS
  - Autor, Fecha de creación, post-condición, pre-condición, de un método en una clase java

Meta Lenguajes

- Lenguaje para describir lenguajes
- Ejemplos
  - Español + símbolos matemáticos para describir conceptos matemáticos
  - EBNF (Extended Backus–Naur Form): lenguaje para describir gramáticas de lenguajes libres de contexto.

Programa de Hilbert

- 1920s. The German mathematician David Hilbert (1862-1943) put forward a new proposal for the foundation of classical mathematics.

- The program (From Wikipedia)

    - A formalization of all mathematics
    - Completeness: a proof that all true mathematical statements can be proved in the formalism.
    - Consistency: a proof that no contradiction can be obtained in the formalism of mathematics. This consistency proof should preferably use only "finitistic" reasoning about finite mathematical objects.
    - Decidability: there should be an algorithm for deciding the truth or falsity of any mathematical statement.

Gödel's first incompleteness theorem

- The theorem from wikipedia:

  "For any consistent formal, computably enumerable theory that proves basic arithmetical truths, an arithmetical statement that is true but not provable in the theory can be constructed. That is, any effectively generated theory capable of expressing elementary arithmetic cannot be both consistent and complete. "

Gödel's second incompleteness theorem

- The theorem from wikipedia:

  "For any formal recursively enumerable (i.e. effectively generated) theory T including basic arithmetical truths and also certain truths about formal provability, T includes a statement of its own consistency if and only if T is inconsistent."

Gödel’s note of august 1963

- Gödel himself remarked that it was largely Turing’s work, in particular the "precise and unquestionably adequate definition of the notion of formal system" given in Turing’s 1937 "On computable numbers, with an application to the Entscheidungsproblem," (Turing 1937), which convinced him that his incompleteness theorems refuted the Hilbert program.

Turing

- 1936 escribe el paper “On Computable Numbers, with an Application to Entscheidungsproblem”

- Crea un máquina abstracta de computación

- Muestra que una de estas máquinas (Universal Turing Machine) puede simular todas la otras (repertorios equivalentes)

- Conjetura que el repertorio sería lo que naturalmente se vería como “computable”

- Claro qué esto es computable por los matemáticos, un modelo abstracto en el cerebro de los humanos.

David Deutsch

- Busca teoría del todo no solo basada en teorías reduccionistas (incluye emergentes)
- Física cuántica
- Epistemología
- The theory of computation
- The theory of evolution

David Deutsch II

- Universalidad y límites de la computación
    - La teoría clásica de la computación es obsoleta
    - Ahora la teoría cuántica de la computación
    - Computadores cuánticos usan el multiverso
    - En 1985 prueba que existe un quantum universal computer
    - Ahora trabaja en la teoría del constructor (La teoría del todo)
    - ver: http://constructortheory.org/
        - The fabric of reality
        - The beginning of infinity

Reflexión

Rompiendo la línea entre diseño y uso

- LP tradicionales hay separación entre el rol de diseñador de lenguaje con el del programador

- Existía además una diferencia entre los LP elegantes (e.g., funcionales) y los eficientes (e.g., C++)

Reflexión

- **Reflexión computacional**:
  - Actividad ejecutada por un sistema de computo cuando hace cómputos y modificaciones sobre sus propios cómputos.
- **Lenguajes con reflexión**
  - API para acceso a las abstracciones del lenguaje
  - API se integra con el comportamiento normal de los programas

Implementando Meta Object Protocols

- Reflexión + OO
- Reflexión brinda acceso a abstracciones del lenguaje
- OO brinda flexibilidad por medio de: abstracción, subtipos, herencia, búsqueda dinámica

Meta Object Protocols

- Interfaces que dan acceso al lenguaje
- **Permiten a los usuarios modificar incrementalmente la implementación y comportamiento del lenguaje.**
- Igualmente permiten escribir programas en el lenguaje
- Mecanismo que intenta conciliar elegancia (e.g., LISP) con eficiencia (e.g., C++).

Reflexión en Java

- Conceptos del lenguaje tienen una representación dentro del lenguaje
    - Clase
    - Campo
    - Método
    - Constructor
    - Modificador (e.g., static)
    - Genéricos
    - Tipos
    - Invocación de métodos

Meta Object Protocol en Java

- Raiz de la jerarquía de clases:
    - java.lang.Object
        - Class<?> getClass()

- java.lang.Class<T>
    - Constructor<?>[] getDeclaredConstructors()
    - Field[] getDeclaredFields()
    - Method[] getDeclaredMethods()

Obteniendo objetos Class

- Object.getClass()
  - `Class c= “hola”.getClass();`
  - `byte[] bytes = new byte[1024];`
  - `Class c = bytes.getClass();`
- Sintaxis .class
  - `Class c = boolean.class;  // correct`
- forname
  - `Class c = Class.forName("com.duke.MyLocaleServiceProvider");`
- Otros
  - `Class c = Double.TYPE;`
  - `Class c = javax.swing.JButton.class.getSuperclass();`

Accediendo miembros

Class Methods for Locating Fields

`Class` API	List of members?	Inherited members?	Private members?
`getDeclaredField()`	no	no	yes
`getField()`	no	yes	no
`getDeclaredFields()`	yes	no	yes
`getFields()`	yes	yes	no

Class Methods for Locating Methods

`Class` API	List of members?	Inherited members?	Private members?
`getDeclaredMethod()`	no	no	yes
`getMethod()`	no	yes	no
`getDeclaredMethods()`	yes	no	yes
`getMethods()`	yes	yes	no

Class Methods for Locating Constructors

`Class` API	List of members?	Inherited members?	Private members?
`getDeclaredConstructor()`	no	N/A[1]	yes
`getConstructor()`	no	N/A[1]	no
`getDeclaredConstructors()`	yes	N/A[1]	yes
`getConstructors()`	yes	N/A[1]	no

Tomado de: http://download.oracle.com/javase/tutorial/reflect/class/classMembers.html

Accediendo miembros II

```
printMembers(c.getFields(), “Fields”);

......

private static void printMembers(Member[] mbrs, String s) {
    out.format("%s:%n", s);
    for (Member mbr : mbrs) {
      if (mbr instanceof Field)
        out.format("  %s%n", ((Field)mbr).toGenericString());
      else if (mbr instanceof Constructor)
         out.format("  %s%n", ((Constructor)mbr).toGenericString());
      else if (mbr instanceof Method)
       out.format("  %s%n", ((Method)mbr).toGenericString());
    }
    if (mbrs.length == 0)
       out.format("  -- No %s --%n", s);
    out.format("%n");
```

Invocando métodos

```
public class InvokeMain {
    public static void main(String... args) {
     try {
         Class<?> c = Class.forName(args[0]);
         Class[] argTypes = new Class[] { String[].class };
         Method main = c.getDeclaredMethod("main", argTypes);
         String[] mainArgs = Arrays.copyOfRange(args, 1, args.length);
         System.out.format("invoking %s.main()%n", c.getName());
         main.invoke(null, (Object)mainArgs);

       // production code should handle these exceptions more gracefully
    } catch (ClassNotFoundException x) {
         x.printStackTrace();
    } catch (NoSuchMethodException x) {
         x.printStackTrace();
```

Anotaciones

Un mecanismo de metadata

- Usando reflexión y metadata se pueden hacer varias extensiones al lenguaje
    - Componentes, e.g., EJB
    - Sin embargo el código de plomería es muy oneroso, e.g., archivos de configuración XML, idiomas de programación complicados
    - Las anotaciones son un mecanismo de metadata para los programas Java.
    - Facilitan la creación de frameworks basados en meta-información

Anotaciones en Java

- Proveen información acerca del programa
- Pero no son parte del programa
- En principio no tienen efecto sobre el programa que anotan
- Usos
  - Información al compilador
  - Procesamiento en tiempo de compilación y despliegue
  - Procesamiento en tiempo de ejecución

Ejemplos de anotaciones

```
//Anotación de clase

@Author(

   name = "Benjamin Franklin",

   date = "3/27/2003"

)

class MyClass() { }



//Anotación de método

@SuppressWarnings(value = "unchecked")

void myMethod() { }
```

Anotaciones usadas por compilador

- @Deprecated

- @Overrride

- @SupressWarnings

Creando anotaciones

- Generalmente los frameworks como EJB, Junit proporcionan anotaciones predefinidas y herramientas para procesarlas
- Sin embargo el programador puede crear su propias anotaciones
- Reglas de creación
  - Se declaran como una interface con la notación @interface
  - Cada método define un elemento de la anotación
  - Método no debe tener parámetros ni cláusulas *throws*
  - Tipos de retorno restringidos a: primitivos, String, Class, enums, annotations y arreglos de estos tipos
  - Los métodos pueden tener valores por defectos

Ejemplo: declaración y uso de una anotación

- Declaración

```
public @interface RequestForEnhancement {

    int    id();

    String synopsis();

    String engineer() default "[unassigned]";

    String date();    default "[unimplemented]";

}
```

- Uso

```
@RequestForEnhancement(

    id       = 2868724,

    synopsis = "Enable time-travel",

    engineer = "Mr. Peabody",

    date     = "4/1/3007"

)

public static void travelThroughTime(Date destination) { ... }
```

Anotaciones marker

- Anotaciones sin elementos se llaman marker

```
public @interface Preliminary { }
```

- En el uso se puede omitir el parentesis

```
@Preliminary public class TimeTravel { ... }
```

Anotaciones de un solo elemento

- El elemento puede ser llamado *value*

```
public @interface Copyright {

    String value();

}
```

- Si se uso *value*, se puede omitir el nombre y el signo =

```
@Copyright("2002 Yoyodyne Propulsion Systems")

public class OscillationOverthruster { ... }
```

Meta anotaciones

- Anotaciones hechas sobre las anotaciones
- Ejemplo: Anotaciones para un framework de test

```
@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

public @interface Test { }
```

- @Retention(RetentionPolicy.RUNTIME): anotación retenida por la VM
- @Target(ElementType.METHOD): Indica que la anotación se aplica a métodos

Procesando anotaciones

- Anotaciones son representadas por el API de reflexión
- Ejemplo: Una herramienta de test

```
public class RunTests {

   public static void main(String[] args) throws Exception {

      int passed = 0, failed = 0;

      for (Method m : Class.forName(args[0]).getMethods()) {

         if (m.isAnnotationPresent(Test.class)) {

            try {

               m.invoke(null);

               passed++;

            } catch (Throwable ex) {

               System.out.printf("Test %s failed: %s %n", m, ex.getCause());

               failed++;

   }}}

   System.out.printf("Passed: %d, Failed %d%n", passed, failed);

}}
```

Procesando anotaciones II

- ¿Cúal es la salida si pasamos este programa?

```
public class Foo {

    @Test public static void m1() { }

    public static void m2() { }

    @Test public static void m3() {

        throw new RuntimeException("Boom");

    }

    public static void m4() { }

    @Test public static void m5() { }

    public static void m6() { }

    @Test public static void m7() {

        throw new RuntimeException("Crash");

    }

    public static void m8() { }

}
```

POJO

POJO

- Acrónimo que significa Simple y viejo Objeto Java (Plain Old Java Object)
- Un POJO tiene
  - Campos
  - Métodos de acceso a estos campos
  - Otros métodos para hacer otras acciones
- Pueden estar anotados para soportar despliegue en frameworks que extienden Java, e.g., JEE

Inversión de Control (IoC)

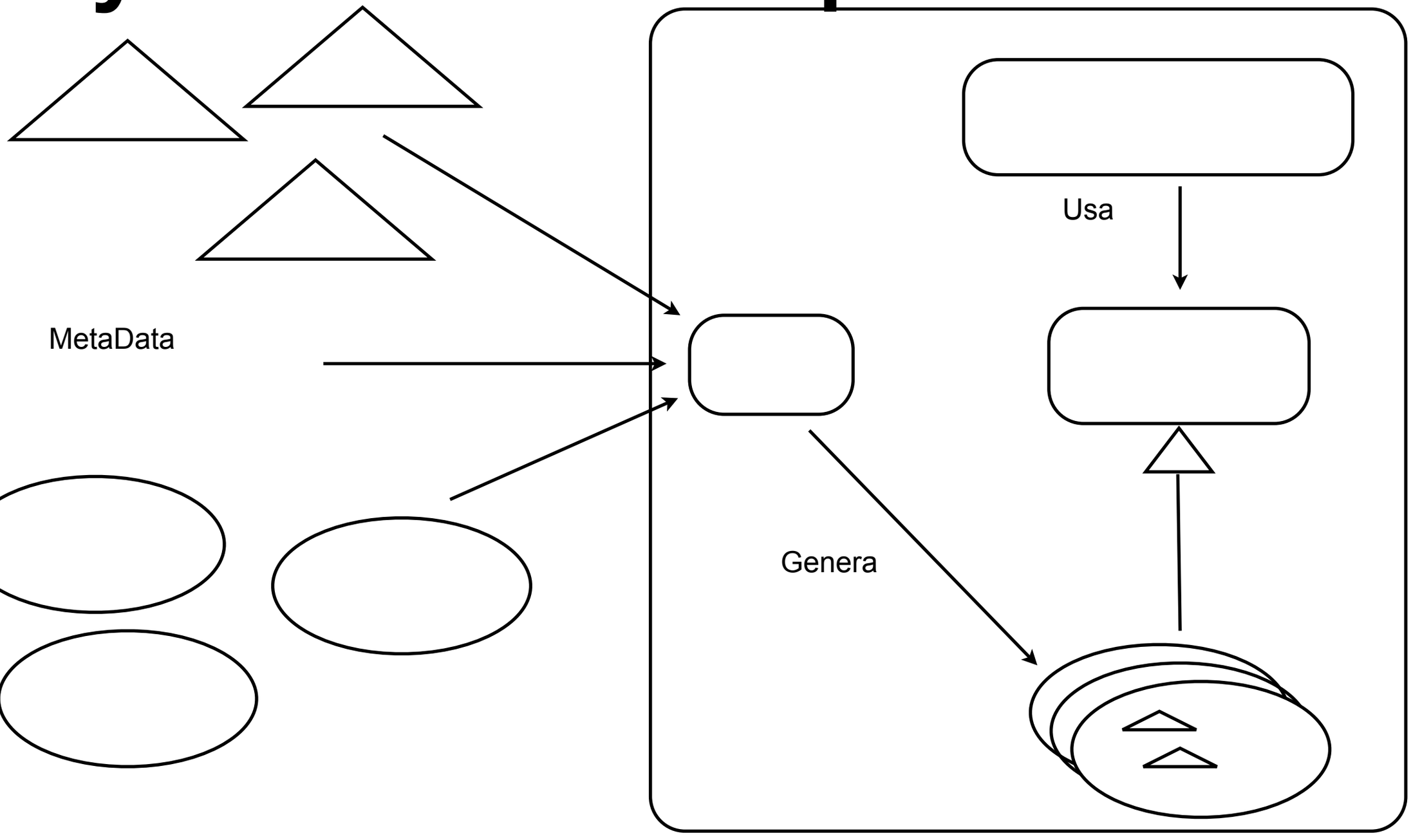
Inversión de Control (IoC)

- Patrón de diseño de software
- El flujo de un sistema es invertido con respecto a la secuencia tradicional de llamados a procedimientos
- En general el flujo es delegado a un elemento fuera del control del usuario
- El usuario desarrolla para un modelo/framework específico, y otra entidad crea el flujo correcto de llamados
- la programación es declarativa

Ejemplos de IoC

- “No nos llame nosotros los llamamos”
- MVC a la Smalltalk
- Inyección de Dependencias en contenedores, e.g., Spring
- Gang Of Four: Method Template
- Frameworks basados en anotaciones, e.g., Junit, JEE

Arquitectura de IoC con Inyección de dependencias



Test de conocimientos

1. Defina MetaData, MetaLenguaje
2. ¿Qué relación tienen Hilbert, Gödel, Turing?
3. ¿Qué busca David Deutsch?
4. ¿Qué es reflexión en los lenguajes de programación?
5. ¿Qué es un Meta Object Protocol?
6. Describa el patrón IoC
7. ¿Qué rol Juegan las anotaciones en una arquitectura de un framework basado en IoC con Inyección de dependencias?

Luis Daniel Benavides Navarro

dnielben@gmail.com