Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Προηγμένες Μέθοδοι Προγραμματισμού

ΠΜΣ 2022-23 (M135.CS1E, M135.CS23B, M135.IC1E, παλαιό: M117)

Meta-Programming

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Meta-programming

- The ability to write programs that treat other programs as their input.
- Write programs that read, generate, analyze, transform and modify other programs.

Why should I wanna do this?

- Program analysis and verification
- Software testing and validation
- Code optimization
- Code generation
- Assist software development (e.g dev tools & IDEs)
- Change the behavior of existing code (e.g. extend legacy code)
- Deal with cross-cutting concerns
- Evolution and adaptiveness

Program analysis

Reason about a specific property of a program (e.g. correctness, security, safety, etc.).

- Static analysis: without actually executing the program (e.g. control-flow or data-flow analyses).
- Dynamic analysis: while executing it (e.g. software testing or monitoring).
- Hybrid: combine both techniques.

Compile-time meta-programming

Interact with the compiler

- directly, through "compiler plugins"
- indirectly, through macros or annotations

Runtime meta-programming

Interact with the running program, as it runs

- through Reflection
- through a Metaobject protocol
- through the language itself (e.g. dynamic languages)

Meta-programming techniques discussed in this class

- Reflection
- Proxies
- Aspects & Aspect-oriented programming
- Metaobject protocols
- Annotations

Metaobject protocols

Gregor Kiczales, Jim des Rivieres, Daniel G. Bobrow, "The Art of the Metaobject Protocol", 1991, MIT Press, ISBN 0-262-61074-4

What is a metaobject protocol?

- A vocabulary (protocol) to access and manipulate the structure and behaviour of systems of objects.
- Typical functions of a metaobject protocol include:
  - Create or delete a new class
  - Create a new property or method
  - Cause a class to inherit from a different class ("change the class structure")
  - Generate or change the code defining the methods of a class

Metaobject protocols and the SOLID principles

- Remember SOLID?
- O in SOLID is the the open/closed principle: software "entities" should be open for extension but closed for modification.
- Metaobject protocols violate it.

Metaobject protocols and AOP

- A metaobject protocol is a way to implement Aspect-oriented Programming.
- AOP is a technique to inject/modify code (advice) at certain points in the execution of a program (pointcuts) in order to deal with cross-cutting concerns (tbd).

Metaclass

- The most widespread realization of a Metaobject protocol in mainstream PLs.
- Supported by Python, Ruby, Groovy, Objective-C (among others) and also by RDF and UML.

Example (Groovy)

```
Number.metaClass.isGreaterThan { Number n -> return delegate > n }
Number.metaClass.isTheAnswer() { ->return delegate == 42 }
println(5.isGreaterThan(3)) //prints true
println(5.isTheAnswer()) //prints false
```

Example (Python)

```
Person = type('Person', (), dict(name='Name', surname='Surname'))
//or equivalently
Person = type('Person', (), {'name':'Name', 'surname':'Surname'})
p = Person()
print(p.surname + ", " + p.name) //prints Surname, Name
```

Annotations

- A simple meta-programming technique
- We will discuss about Java & Groovy annotations

Java annotations

Java Tutorial

Combine annotations with reflection

In the cases where the annotation is not strictly for documentation purposes, use
a runtime retention policy to allow the annotation to be available at runtime.
Then use Reflection to make use of it.

Example - The Jackson JSON Encoder/Decoder

Jackson at a glance

Jackson Annotations

Groovy annotations

They rely on AST (Abstract Syntax Tree) transformations, a powerful compile-time meta-programming idiom of Groovy

Groovy compile-time meta-programming

The Groovy transform package