INF 102
ANALYSIS OF PROG LANGS
REFLECTION

Instructors: James Jones

Outline

- **History and Background**
  - The meta-circular interpreter
- Definitions
- Reflection as used in
  - PHP
  - Ruby
  - Java
- Concerns to keep in mind while using Reflection.
- Practical applications.

"Follow effective action with quiet reflection. From the quiet reflection will come even more effective action."

Peter F. Drucker

History

- It arose naturally in artificial intelligence, where it is intimately linked to the end goal itself.

- Reflection is viewed as the emergent property responsible, at least in part, for what is considered an “intelligent behaviour”.

- Reflection helps us master new skills, cope with incomplete knowledge, define terms, examine assumptions, review our experiences, plan, check for consistency, and recover from mistakes.

- Key strategy for meta-programming.

History

- Languages like LISP had inherent reflective properties.
  - The powerful ‘quote’ mechanism in LISP, Scheme etc enabled code to be treated as data – primitive manifestations of reflection.
- Brian Cantwell Smith's work in the 80s
  - Formalized the concept of reflection
  - Developed two dialects of Lisp namely 2-Lisp and 3-Lisp
  - Became famous in the functional domain and therefore inspired much work there.
- By the end of the 90s- the need for structuring mechanisms was noticed
  - The object-oriented paradigm imposed on itself to take up this challenge.
  - However they were also influenced by the Lisp community

Meta-circular interpreter

- A **self-interpreter**, or **metainterpreter,** is a programming language interpreter written in the language it interprets
- A **meta-circular interpreter** is a special case of a **self-interpreter** in which the existing facilities of the parent interpreter are directly applied to the language being interpreted, without any need for additional implementation
  - Primarily in homoiconic languages

Homoiconicity

- primary representation of programs is also a data structure in a primitive type of the language itself
  - internal and external representations are essentially the same
- *homo* = the same
  icon = representation

- Examples: Lisp, Scheme, R, Mathematica
- Counter-examples: Java, C, Python…
  - Programs are strings, text

Homoiconicity

- (* (sin 1.1) (cos 2))
  >> -0.37087312359709645

Literal. Means:
“don’t interpret me!”

- ` (* (sin 1.1) (cos 2))
  >> `(* (sin 1.1) (cos 2))

- (eval `(* (sin 1.1) (cos 2)))
- >> -0.37087312359709645

What is Eval?

- Way back from McCarthy's paper on LISP
- To a first approximation,
  eval is the exposure of the interpreter itself to the programmer

- In homoiconic languages, eval takes an expression of the language and interprets it
- In non-homoiconic languages, eval takes a string, parses it, and interprets the resulting expression

Where have you seen eval?

- JavaScript

```
<script type="text/javascript">

eval("x=10;y=20;document.write(x*y)");
document.write("<br />" + eval("2+2"));
document.write("<br />" + eval(x+17));

</script>
```

- Python

```
>>> x = 1
>>> print eval('x+1')
2
```

Eval = Evil ?

- discuss

Outline

- History and Background
  - The meta-circular interpreter
- **Definitions**
- Reflection as used in
  - PHP
  - Ruby
  - Java
- Concerns to keep in mind while using Reflection.
- Some practical applications.

Definitions

- General definition of reflection by Brian Smith in the 80s

  "An entity's integral ability to represent, operate on, and otherwise deal with its self in the same way that it represents, operates on and deals with its primary subject matter."

- In programming languages the incarnation of this definition is something like

  "Reflection is the ability of a program to manipulate itself as data during execution."

Definitions

- ***Reification*** - mechanism for encoding execution state as data; providing such an encoding is called reification.
  - Reification is the process by which a user program or any aspect of a programming language that was implicit in the translated program and the run-time system, are expressed in the language itself.
  - This process makes the program available to the program, which can inspect all these aspects as ordinary data .
  - Reification data is often said to be made a first class object.

Intercession

- Modify the characteristics of the elements of the program. Intercession allows you to change the properties of classes, variables, methods, functions, etc. at run-time

Outline

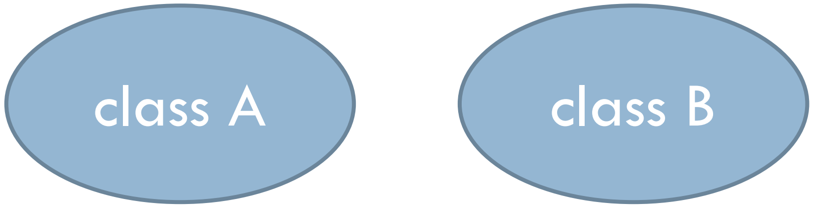
- History and Background
  - The meta-circular interpreter
- Definitions
- **Reflection as used in**
  - **PHP**
  - **Ruby**
  - **Java**
- Concerns to keep in mind while using Reflection.
- Some practical applications.

Reflection in modern PLs

Program text:

```
class A {…}
class B {…}
Etc.
```

Runtime environment:



Etc.

Web Programming - PHP Reflection

- Reflection is designed to reverse engineer various parts of PHP, including classes, functions, and extensions. By "reverse engineer" it means that it gives you all sorts of information that otherwise you would need to try to dig out yourself.

- There are three primary uses for reflection in PHP:

  - You have encoded scripts you need to interact with.
  - The PHP manual isn't wholly up to date and you are unable to, or you don't want to read the source code.
  - You're just curious how something works and would rather not read someone else's PHP .

PHP - Reflection Example

```
<?php
    class myparent {
        public function foo($bar) {
            // do stuff
        }
    }
    class mychild extends myparent {
        public $val;
        private function bar(myparent &$baz) {
            // do stuff
        }
        public function __construct($val) {
            $this->val = $val;
        }
    }
    $child = new mychild('hello world');
    $child->foo('test');
?>
```

PHP - Reflection Example

```
$childreflect = new ReflectionClass('mychild');
echo "This class is abstract: ",
     (int)$childreflect->isAbstract(), "\n";
echo "This class is final: ", (int)$childreflect->isFinal(),"\n";
echo "This class is actually an interface: ",
     (int)$childreflect->isInterface(), "\n";
echo "\$child is an object of this class: ",
     (int)$childreflect->isInstance($child), "\n";
$parentreflect = new ReflectionClass('myparent');
echo "This class inherits from myparent: ",
     (int)$childreflect->isSubclassOf($parentreflect), "\n";
```

```
The output of that is:
This class is abstract: 0
This class is final: 0
This class is actually an interface: 0
$child is an object of this class: 1
This class inherits from myparent: 1
```

Ruby Reflection

- Let's begin with an example : Assume that you want to create a class instance at runtime, and the name of this class depends on the parameter being passed to a function.

- One way to do this is to write conditional loops and create the object. But if there are too many classes then becomes a problem. Solution: Use reflection!

- In Ruby using reflection you can get the following information:
  1. What classes already exist
  2. Information on the methods in those classes
  3. Inheritance etc.

Reflection in Ruby

- ObjectSpace allows us to obtain the reflective information.
- *ObjectSpace.each_object* { |*x*| *puts x* } gives us all the living, non-immediate objects in the process.
- *ObjectSpace.each_object(Class)* { |*x*| *puts x*} gives us all the classes in the ruby process.
- Now the problem becomes easier - Iterate over all the classes, and if the name matches then create an object of that class, and execute the required functions.

Reflection in Ruby

```
class ClassFromString

@@counter = 0
def initialize
@@counter += 1
end
def getCounterValue
puts @@counter
end
end
```

```
def createClassFromString(classname)
  ObjectSpace.each_object(Class) do |x|
    if x.name == classname
      object = x.new
      object.getCounterValue
    end
  end
end

createClassFromString("ClassFromString
```

The above code illustrates the example in code. You can even use superclass method to get the parent name, and construct the entire hierarchy.

Reflection in Java

- **java.lang.Class.**
  - **Class.forName()**

- **Class c = Class.forName(“edu.uci.inf102.Example");**

Reflection in Java

Object.getClass()

```
Class c = foo.getClass();
```

The .class Syntax

```
Class c = boolean.class;

Method m = c.getMethod("add", Integer.TYPE);
Method methodList[] = c.getDeclaredMethods();
Field f = c.getField("sum");
Field fieldList[] = c.getDeclaredFields();
```

Reflection in Java

- **Reflection** is commonly used by programs that require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine. This is a relatively advanced feature and should be used only by developers who have a strong grasp of the fundamentals of the language. Reflection is a powerful technique and can enable applications to perform operations which would otherwise be impossible.

- **Extensibility Features:** An application may make use of external, user-defined classes by creating instances of objects using their fully-qualified names.

Outline

- History and Background
  - The meta-circular interpreter
- Definitions
- Reflection as used in
  - PHP
  - Ruby
  - Java
- **Concerns to keep in mind while using Reflection.**
- Some practical applications.

Performance Penalty

Reflection is powerful, but should not be used indiscriminately. If it is possible to perform an operation without using reflection, then it is preferable to avoid using it. The following concerns should be kept in mind when accessing code via reflection.

Performance Overhead : Because reflection involves types that are dynamically resolved, certain Java virtual machine optimizations can not be performed. Consequently, reflective operations have slower performance than their non-reflective counterparts, and should be avoided in sections of code which are called frequently in performance-sensitive applications.

Security

- **Security Restrictions :** Reflection requires a runtime permission which may not be present when running under a security manager. This is in an important consideration for code which has to run in a restricted security context, such as in an Applet.

- **Exposure of Internals :** Since reflection allows code to perform operations that would be illegal in non-reflective code, such as accessing private fields and methods, the use of reflection can result in unexpected side-effects, which may render code dysfunctional and may destroy portability. Reflective code breaks abstractions and therefore may change behavior with upgrades of the platform.

Outline

- History and Background
  - The meta-circular interpreter
- Definitions
- Reflection as used in
  - PHP
  - Ruby
  - Java
- Concerns to keep in mind while using Reflection.
- **Practical applications.**

Practical Applications of Reflection

Proxies: e.g. a JDK Proxy of a large interface (20+ methods) to wrap (i.e. delegate to) a specific implementation. A couple of methods were overridden using an InvocationHandler, the rest of the methods were invoked via reflection.

Plugins: load specific classes at run-time.

Class Browsers and Visual Development Environments: A class browser has to be able to enumerate members of classes. Visual development environments can benefit from making use of type information available in reflection to aid code development.

Debuggers and Test Tools: Debuggers need to be able to examine private members on classes. Test harnesses can make use of reflection to systematically call a discoverable set APIs defined on a class, to insure a high level of code coverage in a test suite.