Reflection

Idea: introspect and possibly manipulate the structure and behavior of an object
- Compile time
- Runtime

Self-modifying code

Simple example:
- Generate a string
- Load the class that corresponds to the string
- Create an instance of the class

Reflection in Java

- Loading class from a string
  - Class cls = Class.forName(String name);
- Getting the class from an instance
  - Class cls = obj.getClass();

- Getting a constructor
  - Constructor con = cls.getConstructor(Class ... paramTypes);
- Creating an instance
  - Object obj = con.newInstance(Object ... args);

Reflection in Java

- Getting a method
  - Method m = cls.getMethod(String name, Class ... paramTypes);
- Invoking a method
  - m.invoke(Object obj, Object ... args);

- Getting a field
  - Field f = cls.getField(String name);
- Setting a field
  - f.set(Object obj, Object value);
  - setBoolean, setByte, setChar, setShort, setInt, setLong, setDouble, setFloat
Reflection in Java

- Manipulation?
  - f.setAccessible(true)
  - m.setAccessible(true);

- JVMTI
  - RetransformClasses
  - redefineClasses
  - java.lang.instrument

- In Languages like Java, a class is a type schema and the schema cannot be altered at runtime.

Prototype

- Idea: don’t derive objects from classes, derive objects from other objects (prototype objects)
  - Alternative to subtyping, still gives inheritance
  - Usually implemented through delegation

Example: Javascript

```javascript
var o1 = {name: "one", p1: 1, p2: 2};
var o2 = {name: "two", p3: 3};
o2.__proto__ = o1;
```

Prototype and Javascript

- Not all engines expose __proto__

- Alternative example:

```javascript
var o1 = {name: "one", p1: 1, p2: 2};
var o2 = Object.create(o1);
o2.p3 = 3;
```

Prototype and Javascript

- Constructor function
  - function person(name, age) {
    this.name = name;
    this.age = age;
  }
  - var person1 = new person("Alice", 25);
  - var person2 = new person("Bob", 30);
  - person1.job = "security researcher";
  - person1.getJob = function() { return this.job; };
C# Extension Methods

- C# is structurally similar to Java
- How can we add a method to an existing class?
  - Idea: Use a static method of an extension class and allow it to be called like an instance method of another class
    - “Duality” between static method which takes instance as first argument and instance method.
  - Example:
    ```csharp
    using ExtensionMethods;

    string s = "Hello Extension Methods";
    int i = s.WordCount();
    ```

---

Scala: Pimp My Library

- Idea: alter the behavior of libraries through implicit conversions
  - Example:
    ```scala
    val x = Array (1,2,3)
    val y = Array (4,5,6)
    val z = x append y
    implicit def enrichArray[T](xs: Array[T]) = new RichArray[T]
    ```

---

C# Extension Methods

```csharp
namespace ExtensionMethods {
    public static class MyExtensions {
        public static int WordCount(this String str) {
        }
    }
}
```

(Example from MSDN)

- Compiler sees a method call
- Check if there is an instance method
- Check for extension methods in the scope

---

Scala: Pimp My Library

```scala
class RichArray[T](value: Array[T]) {
    def append(other: Array[T]): Array[T] = {
        val result = new Array[T](value.length + other.length)
        Array.copy(value, 0, result, 0, value.length)
        Array.copy(other, 0, result, value.length, other.length) result
    }
}
```

(Example by Martin Odersky)
Metaclasses

- As we have seen in Java, a Class is an Object.
- A Metaclass is a class whose instances are classes.
- A Metaobject protocol allows the programmer to change the behavior of inheritance, method dispatch, and constructors.
- Popular examples: Scheme and Common Lisp (CLOS)

Metaclasses in Python

```python
def newfunc(obj):
    print 'the new method has been called'

class MyExtender(type):
    def __new__(meta, classname, supers, classdict):
        classdict['newmethod'] = newfunc
        return type.__new__(meta, classname, supers, classdict)

class MyNewClass(MyClass):
    __metaclass__ = MyExtender
```

Aspect-Oriented Programming

- Problem: Some elements in programming seem to be cross-cutting concerns.
  - E.g., Logging, Security, Tracing
- Idea: Factor cross-cutting concerns out into Aspects and weave them into the code.
  - Most common: at compile time or at load time.
  - Also possible: at runtime.

Aspect

- Advice
  - Encapsulation of a unit of code and a methodology to apply it
- Pointcut
  - Definition where an advice can be applied

AspectJ

```java
public aspect BandwidthTracing {
    pointcut sendingData() : call(int SocketChannel.write(ByteBuffer));
    pointcut receivingData() : call(int SocketChannel.read(ByteBuffer));

    after() returning(int written) : sendingData() {
        if (written != 0) {
            PerformanceTracer.addToCumulativeTraceData("NETWORK", "SENT_DATA", written);
        }
    }

    after() returning(int read) : receivingData() {
    }
```
IOC

- Problem: In traditional OOP dependencies are statically created through assignments. This interleaves business logic and configuration.
  - Example: database interface and its wiring to app server logic
- Idea: The Hollywood Principle
  - Don’t call us, we call you
  - Binding gets injected at runtime by the container
  - Configuration tells the container what to inject where
- Popular: Dependency Injection
  - Spring Framework, Google Guice

Example: Google Guice

```java
public class MyClass {
    private final Logger logger;
    private final DBConnectionFactory connFactory;

    @Inject
    MyClass(final Logger logger, final DBConnectionFactory connFactory) {
        this.logger = logger;
        this.connFactory = connFactory;
    }
}
```

```
public class MyModule extends AbstractModule {
    protected void configure() {
        bind(Logger.class, Log4JLogger.class);
        bind(DBConnectionFactory.class, DB2ConnectionFactory.class);
    }
}
```

Injector injector = Guice.createInjector(new MyModule());
MyClass cls = injector.getInstance(MyClass.class);
```