Groovy

Metaobject protocol

Objects in Groovy

- In a Groovy application we’ll work with three kinds of objects: POJOs, POGOs and Groovy interceptors.
  - Plain old Java objects (POJOs) are regular Java objects
    - We can create them using Java or other languages on the Java Virtual Machine (JVM).
  - Plain old Groovy objects (POGOs) are classes written in Groovy.
    - They extend *java.lang.Object* but implement the *groovy.lang.GroovyObject* interface.
  - Groovy interceptors are Groovy objects that extend *GroovyInterceptable* and have a method-interception capability

GroovyObject interface

```
//This is an excerpt of GroovyObject.java from Groovy source code
package groovy.lang;
public interface GroovyObject {
  Object invokeMethod(String name, Object args);
  Object getProperty(String property);
  void setProperty(String property, Object newValue);
  MetaClass getMetaClass();
  void setMetaClass(MetaClass metaClass);
}
```

- *invokeMethod()*, *getProperty()* and *setProperty()* make Groovy objects highly dynamic.
  - We can use them to work with methods and properties created on the fly.
- *getMetaClass()* and *setMetaClass()* make it very easy to create proxies to intercept method calls on POGOs, as well as to inject methods on POGOs.
- Once a class is loaded into the JVM, we can't change the metaobject Class for it.
  - We can change its MetaClass by calling *setMetaClass()*.
    - This gives us a feeling that the object changed its class at runtime.

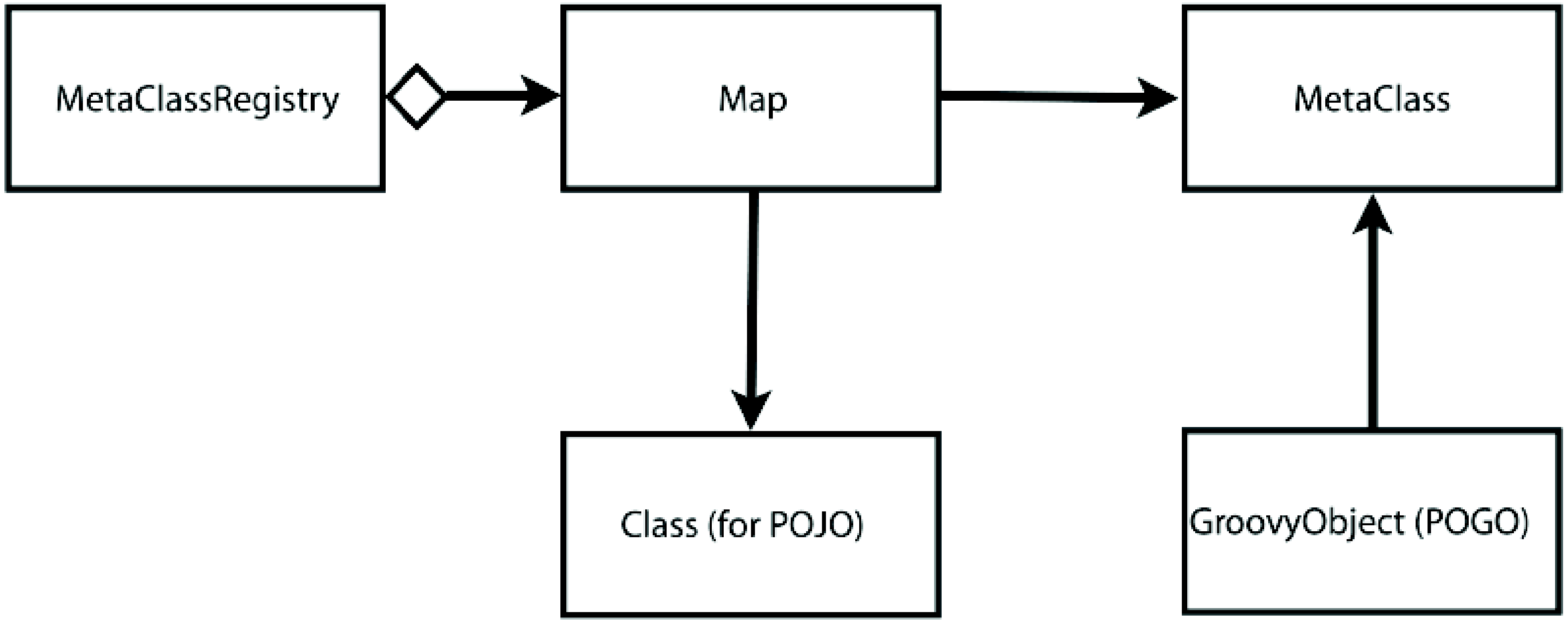
GroovyInterceptable interface

```
//This is an excerpt of GroovyInterceptable.java from Groovy source code
package groovy.lang;
public interface GroovyInterceptable extends GroovyObject {
}
```

- It’s a marker interface that extends *GroovyObject*
- All method calls - both existing methods and nonexistent methods - on an object that implements this interface are intercepted by its *invokeMethod()*.

Grovy metaprogramming

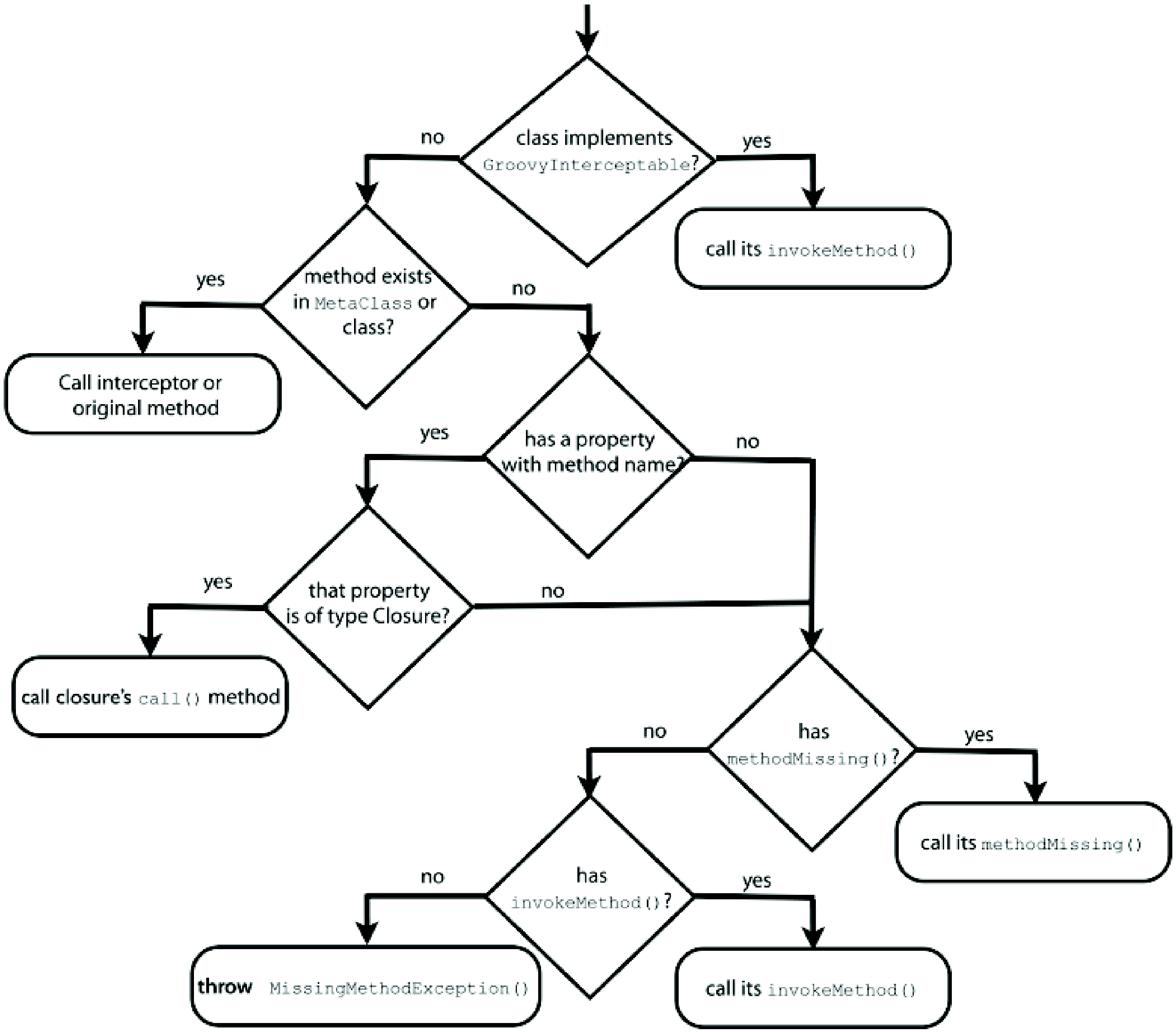
- Groovy allows metaprogramming for POJOs and POGOs
- For POJOs, Groovy maintains a MetaClassRegistry class of MetaClasses
- POGOs have a direct reference to their MetaClass.



Method handling for POJOs

- For a POJO, Groovy fetches its MetaClass from the application-wide MetaClassRegistry and delegates method invocation to it.
- Any interceptors or methods we’ve defined on its MetaClass take precedence over the POJO’s original method.

Metod handling for POGOs



class implements GroovyInterceptable?
no
yes
call its invokeMethod()
method exists in MetaClass or class?
yes
no
Call interceptor or original method
has a property with method name?
yes
no
that property is of type Closure?
yes
no
call closure's call() method
has methodMissing()?
no
yes
call its methodMissing()
has invokeMethod()?
no
yes
throw MissingMethodException()
call its invokeMethod()

Intercepting Methods Using MOP

- In Groovy we can implement aspect-oriented programming (AOP) - such as

  method interception or method advice—fairly easily.
- There are three types of advice.
  - The before advice is code for a concern we’d want to execute before a certain operation.
  - The after advice is executed after an operation’s execution.
  - The around advice is executed instead of the intended operation.
- We can use MOP to implement these advice types or interceptors.

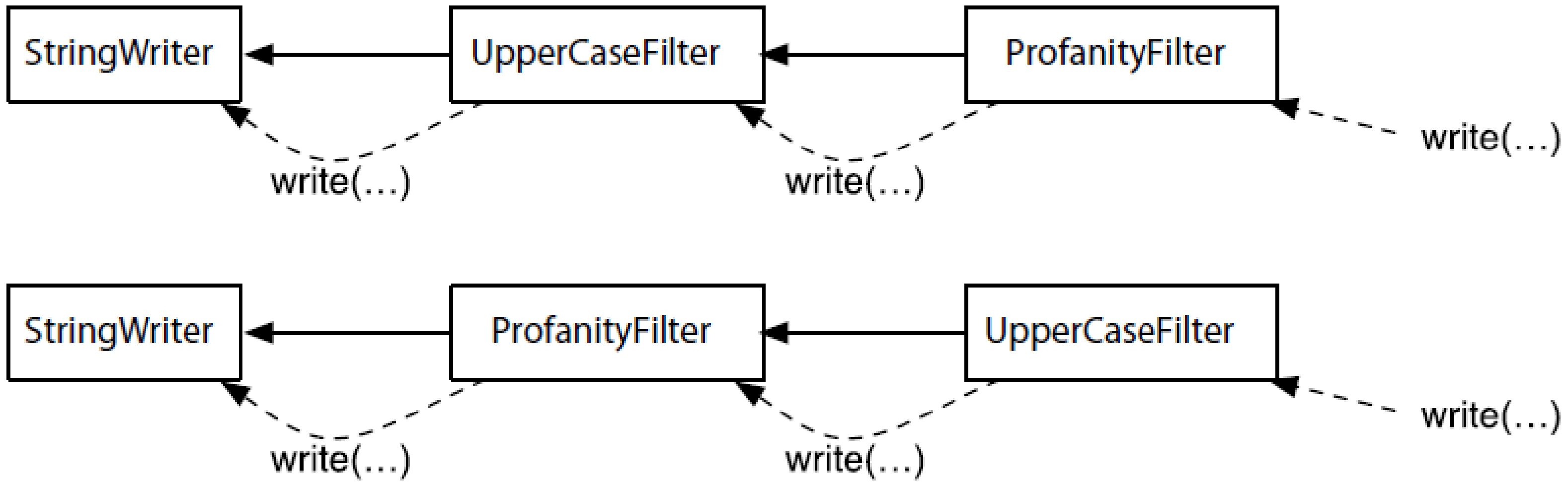
Intercepting Methods Using *GroovyInterceptable*

- If a Groovy object implements *GroovyInterceptable*, then its *invokeMethod()* is called when any of its methods are called
  - both existing methods and nonexistent methods.
- *GroovyInterceptable*’s *invokeMethod*() hijacks all calls to the object.
  - If we want to perform an around advice, we simply implement our logic in this method, and we’re done.
  - If we want to implement the before or after advice (or both), we first implement our before/after logic, then route the call to the actual method at the appropriate time.

MOP Method Injection

- Using Groovy’s MOP, we can inject behavior using any of the following:
  - Categories
  - *ExpandoMetaClass*
  - Mixins

Filter chaining


StringWriter
UpperCaseFilter
ProfanityFilter
write(…)
write(…)
write(…)
StringWriter
ProfanityFilter
UpperCaseFilter
write(…)
write(…)
write(…)