Runtime class generation

let the machine work for you

autumn 2016

Jan @Novoj Novotný

Follow presentation with me at

http://novoj.github.io/reveal.js/runtime-class-generation.html

Code with me

git@github.com:novoj/RuntimeClassGeneration.git

About me

- Java developer at FG Forrest
- YouTuber at Kafemlejnek.TV
- blogger at blog.novoj.net
- jOpenspace non-conference co-organizer
- co-author of MonkeyTracker service
- occasional speaker, father,
  MTB rider, runner ...

Wouldn't it be nice if this code

```
public interface PersonDao {

    void create(String firstName, String lastName, LocalDate birthDate);

    void store(Person person);

    Person getById();

    boolean removeById();

}
```

was automatically implemented by machine?

It's already out there

Just to name a few:

- Spring data
- MyBatis
- Active Record (JRuby)

Spring Data Example

Example 13. Query creation from method names

```
public interface PersonRepository extends Repository<User, Long> {

  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

But how they do it?

Using automatic class (code) generation in:

compile time

- AspectJ
- ByteBuddy
- BTrace
- Lombok

runtime

- JdkProxy
- CgLib
- Javassist
- ByteBuddy

Both approaches have their pros and cons.

Project Lombok

Person.java | Family.java

```
package com.ociweb.jnb.lombok;

import java.util.Date;

import lombok.Data;

@Data
public class Person {
    private final String firstName;
    private final String lastName;
    private String address;
    private String city;
    private String state;
    private String zip;
    private final Date dateOfBirth;
}
```

Outline

- com.ociweb.jnb.lombok
- import declarations
  - java.util.Date
  - lombok.Data
- Person
  - Person(String, String, Date)
  - getFirstName() : String
  - getLastName() : String
  - getAddress() : String
  - setAddress(String) : void
  - getCity() : String
  - setCity(String) : void
  - getState() : String
  - setState(String) : void
  - getZip() : String
  - setZip(String) : void
  - getDateOfBirth() : Date
  - equals(Object) : boolean
  - hashCode() : int
  - toString() : String
  - firstName : String
  - lastName : String
  - address : String
  - city : String
  - state : String
  - zip : String
  - dateOfBirth : Date

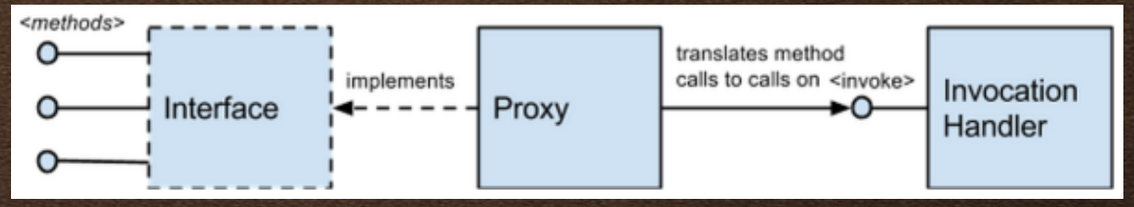
Lombok Person Example / POM Example

Let's prototype!

Branch: tutorial01

Look at:

- Person
- Property Accessor
- Actual memory model
- Proxy implementation

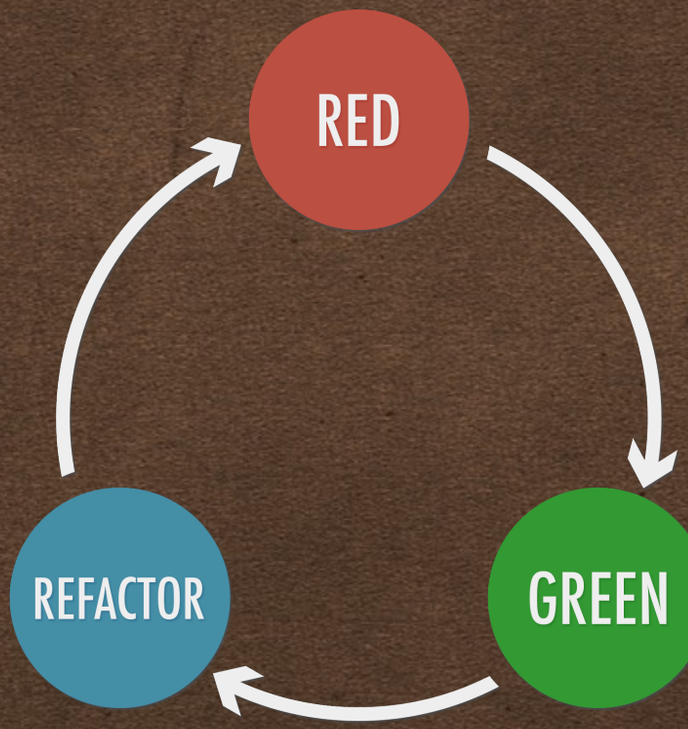


Pros

- type safe code around hashmap
- supports refactoring
- supports searching

Cons

- string parsing - method.getName()
- ugly code
- most likely pretty slow

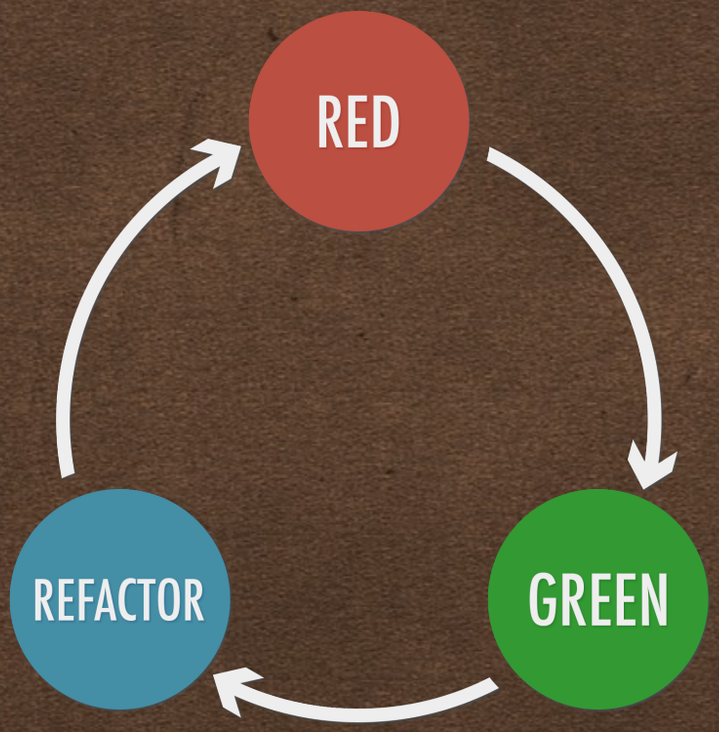


Refactor!

Proxy class caching

Branch: tutorial02

- java.lang.reflect.Proxy already caches classes
- but we can do this more effectively
- this is valid even for other class generators - I recommend disabling generator caching and implement your own



Refactor!

Method logic caching

Branch: tutorial03

Let's make it even better:

1. convert IF to lambda functions
2. cache pair Proxy.method + service lambda
3. cache information about method name decomposition, so that parsing is done only once

Look at:

- Concrete usage
- Dispatcher handler
- Method classification

Custom method body

Branch: tutorial04

What if we need custom logic on a proxy?

What if we want to mix and match custom logic with automatic one?

Default / abstract methods to the rescue.

Look at:

- Aging person trait
- Customized person contract
- Test suite
- Updated dispatcher handler

DAO implementation

all building blocks in place

Branch: tutorial05

Nothing much new here

Look at:

- Dao interface
- Person Dao "implementation"
- Test suite
- Memory repository
- Dao method implementation

DAO implementation

nicer create & add method

Branch: tutorial06

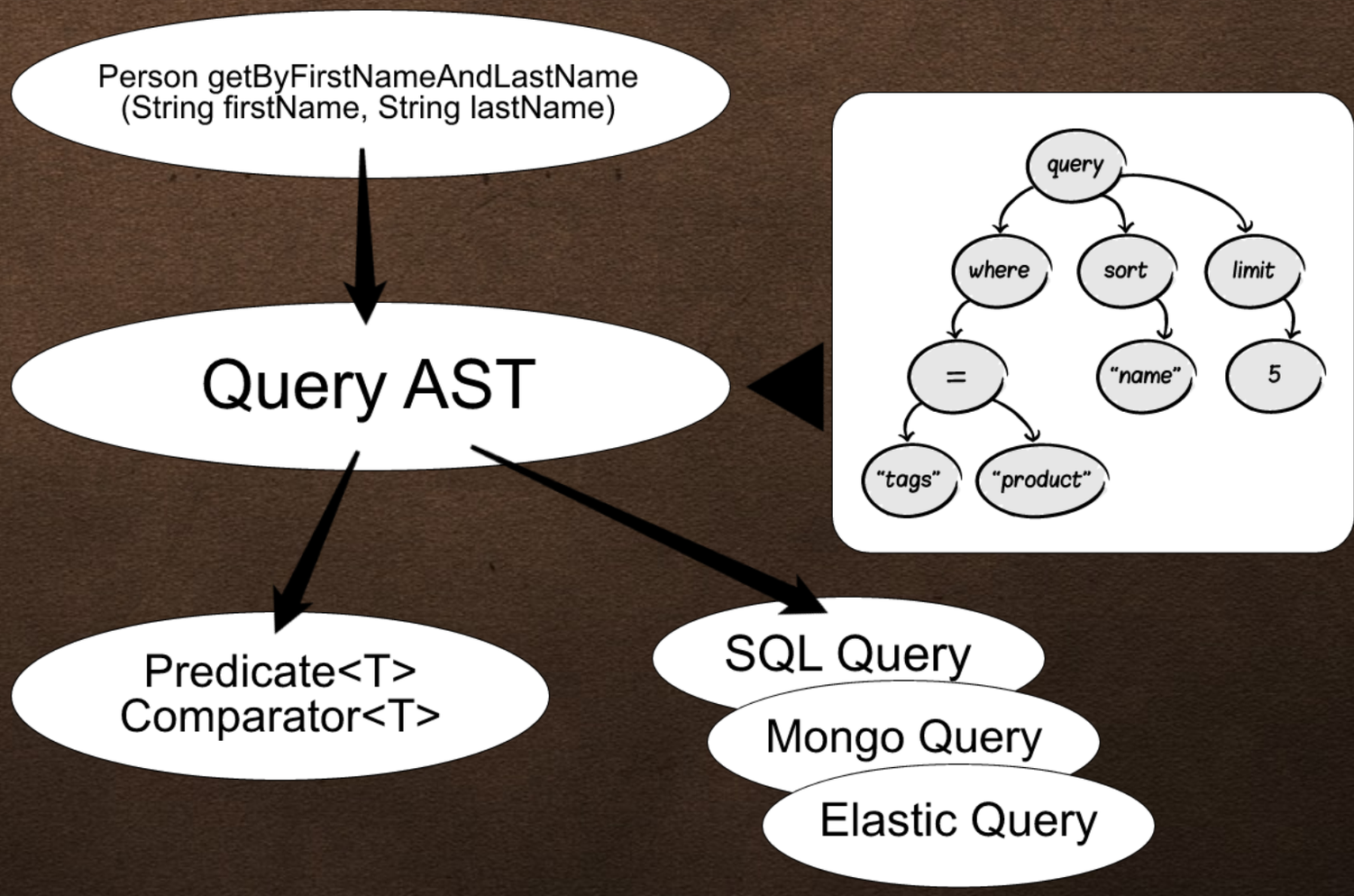
Let's get advantage of parameters!

Look at:

- Person Dao "implementation"
- Test suite
- Add method executor
- Updated proxy generator

DAO implementation

finder / removal implementation



DAO implementation

finder implementation

Branch: tutorial07

- C CREATE
- R READ
- U UPDATE
- D DELETE

Look at:

- Person Dao "implementation"
- Updated proxy generator
- Test suite
- Get method executor
- Remove method executor
- Query AST
- Keywords (reserved words)

Bonus

proxy serialization

Branch: master

We can't simply serialize proxy - class might not be known at the moment of deserialization.

We need to serialize "recipe" how to reconstruct the class.

Look at:

- Serializable Proxy
- Test suite

Bonus

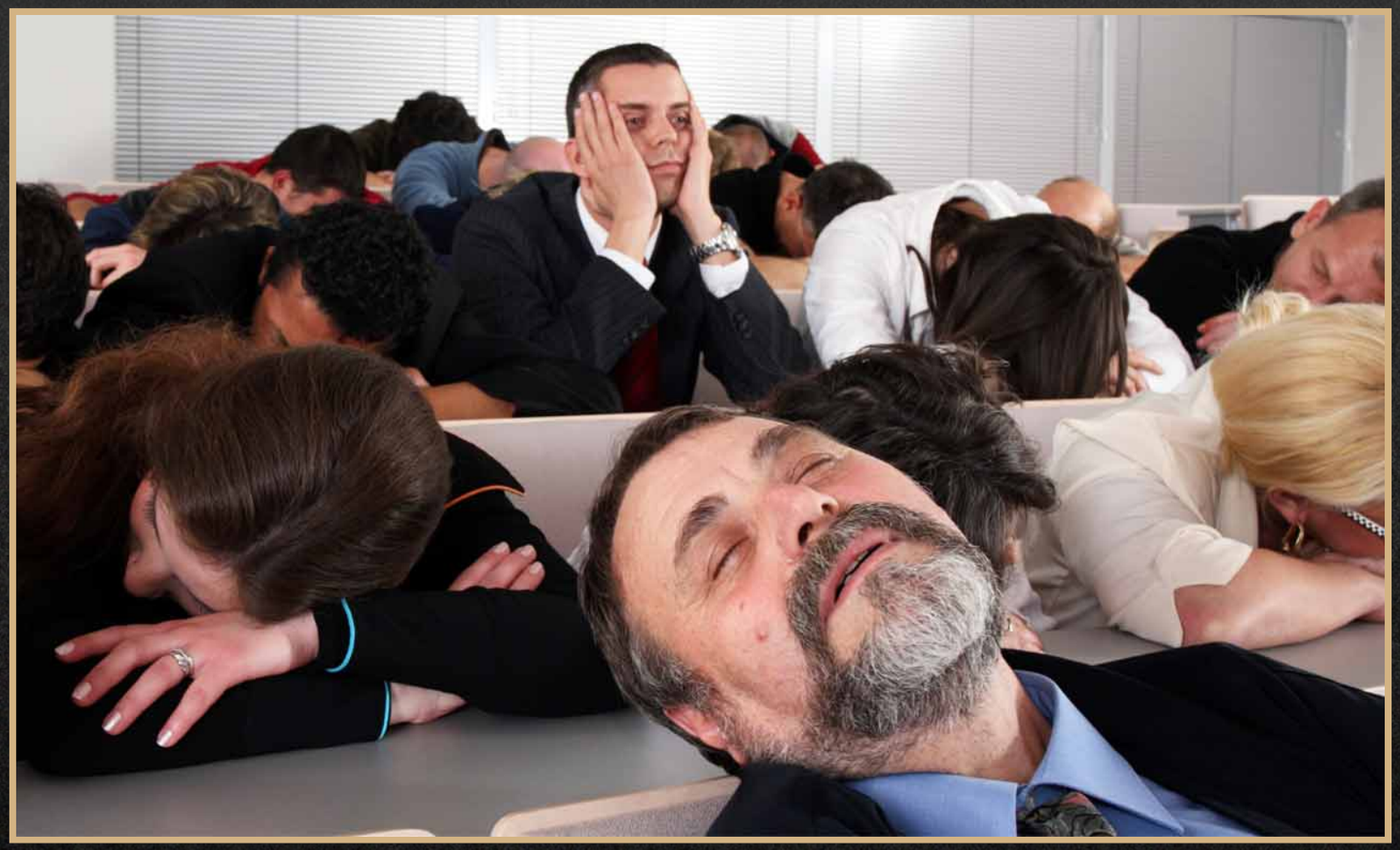
always repackage

```
<plugin>
    <groupid>org.sonatype.plugins</groupid>
    <artifactid>jarjar-maven-plugin</artifactid>
    <version>1.9</version>
    <executions>
        <execution>
                <phase>package</phase>
                <goals>
                        <goal>jarjar</goal>
                </goals>
                <configuration>
                        <overwrite>true</overwrite>
                        <includes>
                                <include>org.javassist:javassist</include>
                        </includes>
                        <rules>
                                <rule>
                                        <pattern>javassist.**</pattern>
                                        <result>cz.novoj.generation.internal.javassist.@1</result>
                                </rule>
                        </rules>
                </configuration>
        </execution>
    </executions>
</plugin>
```

Bonus

additional topics

- annotations on methods are not inherited in Java and some class generations utils doesn't copy them on overriden methods on subclass, look what Spring guys worked around this
- generics resolution - is tricky, again see Spring implementation
- performance of proxies is not so bad, but dynamic nature makes them much slower than static class implementation



Thank your for your attention

Contact me at @Novoj or novotnaci@gmail.com