How can I eliminate duplicated Enum code?

Asked 16 years, 3 months ago Modified 4 years, 11 months ago Viewed 9k times

24

I have a large number of Enums that implement this interface:

```
/**
 * Interface for an enumeration, each element of which can be uniquely
identified by its code
 */
public interface CodableEnum {

    /**
     * Get the element with a particular code
     * @param code
     * @return
     */
    public CodableEnum getByCode(String code);

    /**
     * Get the code that identifies an element of the enum
     * @return
     */
    public String getCode();
}
```

A typical example is:

```
public enum IMType implements CodableEnum {

    MSN_MESSENGER("msn_messenger"),
    GOOGLE_TALK("google_talk"),
    SKYPE("skype"),
    YAHOO_MESSENGER("yahoo_messenger");

    private final String code;

    IMType (String code) {
        this.code = code;
    }

    public String getCode() {
        return code;
    }

    public IMType getByCode(String code) {
        for (IMType e : IMType.values()) {
            if (e.getCode().equalsIgnoreCase(code)) {
                return e;
            }
        }
    }
}
```

As you can imagine these methods are virtually identical in all implementations of CodableEnum. I would like to eliminate this duplication, but frankly don't know how. I tried using a class such as the following:

```
public abstract class DefaultCodableEnum implements CodableEnum {

    private final String code;

    DefaultCodableEnum(String code) {
        this.code = code;
    }

    public String getCode() {
        return this.code;
    }

    public abstract CodableEnum getByCode(String code);
}
```

But this turns out to be fairly useless because:

1. An enum cannot extend a class
2. Elements of an enum (SKYPE, GOOGLE_TALK, etc.) cannot extend a class
3. I cannot provide a default implementation of getByCode(), because DefaultCodableEnum is not itself an Enum. I tried changing DefaultCodableEnum to extend java.lang.Enum, but this doesn't appear to be allowed.

Any suggestions that do not rely on reflection? Thanks, Don

java enums enumeration

Share
Improve this question
Follow

edited Jan 8, 2020 at 21:20

Noumenon
6,372 ● 4 ● 62 ● 81

asked Sep 16, 2008 at 21:15

Dónal
187k ● 176 ● 581 ● 843

15 Answers

Sorted by: Highest score (default)

14

You could factor the duplicated code into a `CodeableEnumHelper` class:

```
public class CodeableEnumHelper {
    public static CodeableEnum getByCode(String code, CodeableEnum[] values) {
        for (CodeableEnum e : values) {
            if (e.getCode().equalsIgnoreCase(code)) {
                return e;
            }
        }
```

```
        return null;
    }
}
```

Each `CodeableEnum` class would still have to implement a `getByCode` method, but the actual implementation of the method has at least been centralized to a single place.

```
public enum IMType implements CodeableEnum {
    ...
    public IMType getByCode(String code) {
        return (IMType)CodeableEnumHelper.getByCode(code, this.values());
    }
}
```

Share

Improve this answer

Follow

edited Jan 30, 2015 at 2:51

varzeak

150 • 7

answered Sep 16, 2008 at 23:22

dave

999 • 5 • 8

1 As an improvement, you could just eliminate the getByCode from the interface. It's enough to provide it in the helper (as a static method). That way there's even less duplication in the individual enums. – sleske Mar 18, 2010 at 16:46

this.values() referes to a static method and should be IMType.values(). I would even remove the requirement to pass in an array of value an instead pass in an enum class literal allowing to use type.getEnumConstants(). – whiskeysierra Jul 31, 2010 at 10:09

1 typo: CodableEnum should be CodeableEnum – Antony Stubbs Aug 16, 2012 at 13:18

7

Abstract enums are potentially very useful (and currently not allowed). But a proposal and prototype exists if you'd like to lobby someone in Sun to add it:

http://freddy33.blogspot.com/2007/11/abstract-enum-ricky-carlson-way.html

Sun RFE:

https://bugs.java.com/bugdatabase/view_bug?bug_id=6570766

Share

Improve this answer

Follow

edited Nov 10, 2023 at 19:48

Community Bot

1 • 1

answered Sep 16, 2008 at 21:43

Alex Miller

70.1k • 25 • 124 • 168

5

To tidy up dave's code:

```
public class CodeableEnumHelper {
    public static <E extends CodeableEnum> E getByCode(
        String code, E[] values
```

```
    ) {
        for (E e : values) {
            if (e.getCode().equalsIgnoreCase(code)) {
                return e;
            }
        }
        return null;
    }
}

public enum IMType implements CodableEnum {
    ...
    public IMType getByCode(String code) {
        return CodeableEnumHelper.getByCode(code, values());
    }
}
```

Or more efficiently:

```
public class CodeableEnumHelper {
    public static <E extends CodeableEnum> Map<String,E> mapByCode(
        E[] values
    ) {
        Map<String,E> map = new HashMap<String,E>();
        for (E e : values) {
            map.put(e.getCode().toLowerCase(Locale.ROOT), value) {
        }
        return map;
    }
}

public enum IMType implements CodableEnum {
    ...
    private static final Map<String,IMType> byCode =
        CodeableEnumHelper.mapByCode(values());
    public IMType getByCode(String code) {
        return byCode.get(code.toLowerCase(Locale.ROOT));
    }
}
```

Share

Improve this answer

Follow

edited Oct 11, 2008 at 1:06

answered Sep 17, 2008 at 10:41

Tom Hawtin - tackline

147k ● 30 ● 221 ● 312

I think you need to substitute 'extends' for 'implements' in the code above – Dónal Sep 17, 2008 at 13:41

2

I had a similar issue with a localization component that I wrote. My component is designed to access localized messages with enum constants that index into a resource bundle, not a hard problem.

I found that I was copying and pasting the same "template" enum code all over the place. My solution to avoid the duplication is a code generator that accepts an XML configuration file with the enum constant names and constructor args. The output is the Java source code with the "duplicated" behaviors.

Now, I maintain the configuration files and the generator, not all of the duplicated code. Everywhere I would have had enum source code, there is now an XML config file. My build scripts detect out-of-date generated files and invoke the code generator to create the enum code.

You can see this component here. The template that I was copying and pasting is factored out into an XSLT stylesheet. The code generator runs the stylesheet transformation. An input file is quite concise compared to the generated enum source code.

HTH,
Greg

Share
Improve this answer
Follow

edited Sep 17, 2008 at 16:02

answered Sep 16, 2008 at 21:45

Greg Mattes
33.9k ● 15 ● 76 ● 105

1

Unfortunately, I don't think that there is a way to do this. Your best bet would pro ably be to give up in emums altogether and use conventional class extension and static members. Otherwise, get used to duplicating that code. Sorry.

Share Improve this answer Follow

answered Sep 16, 2008 at 21:24

Daniel Spiewak
55.1k ● 14 ● 111 ● 120

1

Create a type-safe utility class which will load enums by code:

The interface comes down to:

```
public interface CodeableEnum {
    String getCode();
}
```

The utility class is:

```
import java.lang.reflect.InvocationTargetException;
```

```
public class CodeableEnumUtils {
    @SuppressWarnings("unchecked")
    public static <T extends CodeableEnum>  T getByCode(String code, Class<T>
enumClass) throws IllegalArgumentException, SecurityException,
IllegalAccessException, InvocationTargetException, NoSuchMethodException {
        T[] allValues = (T[]) enumClass.getMethod("values", new
Class[0]).invoke(null, new Object[0]);
        for (T value : allValues) {
            if (value.getCode().equals(code)) {
                return value;
            }
        }
        return null;
}
```

}

A test case demonstrating usage:

```
import junit.framework.TestCase;


public class CodeableEnumUtilsTest extends TestCase {
    public void testWorks() throws Exception {
    assertEquals(A.ONE, CodeableEnumUtils.getByCode("one", A.class));
      assertEquals(null, CodeableEnumUtils.getByCode("blah", A.class));
    }

enum A implements CodeableEnum {
    ONE("one"), TWO("two"), THREE("three");

    private String code;

    private A(String code) {
        this.code = code;
    }

    public String getCode() {
        return code;
    }
}
}
```

Now you are only duplicating the getCode() method and the getByCode() method is in one place. It might be nice to wrap all the exceptions in a single RuntimeException too :)

Share Improve this answer Follow

answered Sep 17, 2008 at 0:02

triggerNZ

4,701 ● 4 ● 30 ● 35

Great answer, but I should have specified in my original question, that I'd prefer answers that don't rely on reflection. I've updated the question. – Dónal Sep 17, 2008 at 4:02

No need to use reflection. There is Class.getEnumConstants(). – whiskeysierra Jul 31, 2010 at 10:10

1

Here I have another solution:

```
interface EnumTypeIF {
String getValue();

EnumTypeIF fromValue(final String theValue);

EnumTypeIF[] getValues();

class FromValue {
  private FromValue() {
  }

  public static EnumTypeIF valueOf(final String theValue, EnumTypeIF
theEnumClass) {

    for (EnumTypeIF c : theEnumClass.getValues()) {
      if (c.getValue().equals(theValue)) {
        return c;
      }
    }
    throw new IllegalArgumentException(theValue);
  }
}
```

The trick is that the inner class can be used to hold "global methods".

Worked pretty fine for me. OK, you have to implement 3 Methods, but those methods, are just delegators.

Share

Improve this answer

Follow

edited Nov 26, 2012 at 21:09

Peter O.

32.8k ● 14 ● 84 ● 97

answered Mar 20, 2011 at 11:02

bueyuekt

11 ● 1

0

It seems like you are actually implementing run time type information. Java provides this as a language feature.

I suggest you look up RTTI or reflection.

Share Improve this answer Follow

answered Sep 16, 2008 at 21:28

Dave Hillier

19k ● 10 ● 45 ● 89

Could you elaborate on this? I don't see what you mean. – Mwanji Ezana Sep 16, 2008 at 21:47

0

I don't think this is possible. However, you could use the enum's valueOf(String name) method if you were going to use the enum value's name as your code.

Share Improve this answer Follow

answered Sep 16, 2008 at 21:33

user8681

But where the code is not the same as the Enum's name, that won't work – Dónal Sep 16, 2008 at 22:40

0

How about a static generic method? You could reuse it from within your enum's getByCode() methods or simply use it directly. I always user integer ids for my enums, so my getById() method only has do do this: return values()[id]. It's a lot faster and simpler.

Share Improve this answer Follow

answered Sep 16, 2008 at 21:37

Gabriel

0

If you really want inheritance, don't forget that you can implement the enum pattern yourself, like in the bad old Java 1.4 days.

Share Improve this answer Follow

answered Sep 16, 2008 at 21:53

Mwanji Ezana

924 ● 8 ● 15

0

About as close as I got to what you want was to create a template in IntelliJ that would 'implement' the generic code (using enum's valueOf(String name)). Not perfect but works quite well.

Share Improve this answer Follow

answered Sep 16, 2008 at 22:39

Javamann

2,902 ● 2 ● 25 ● 22

0

In your specific case, the getCode() / getByCode(String code) methods seems very closed (euphemistically speaking) to the behaviour of the toString() / valueOf(String value) methods provided by all enumeration. Why don't you want to use them?

Share Improve this answer Follow

answered Sep 17, 2008 at 11:00

Nicolas

24.7k ● 5 ● 61 ● 67

> Because the codes are stored in the database and the enum name is a symbol only useful to the programmer. At least, that's the way it is around here. – Mr. Shiny and New 安宇 Oct 21, 2008 at 20:13

0

Another solution would be not to put anything into the enum itself, and just provide a bi-directional map Enum <-> Code for each enum. You could e.g. use ImmutableBiMap from Google Collections for this.

That way there no duplicate code at all.

Example:

```
public enum MYENUM{
  VAL1,VAL2,VAL3;
}

/** Map MYENUM to its ID */
public static final ImmutableBiMap<MYENUM, Integer> MYENUM_TO_ID =
new ImmutableBiMap.Builder<MYENUM, Integer>().
put(MYENUM.VAL1, 1).
put(MYENUM.VAL2, 2).
put(MYENUM.VAL3, 3).
build();
```

Share

edited Mar 18, 2010 at 17:37

answered Mar 18, 2010 at 17:06

Improve this answer

Follow

sleske

83.5k • 38 • 194 • 235

0

In my opinion, this would be the easiest way, without reflection and without adding any extra wrapper to your enum.

You create an interface that your enum implements:

```
public interface EnumWithId {

    public int getId();

}
```

Then in a helper class you just create a method like this one:

```
public <T extends EnumWithId> T getById(Class<T> enumClass, int id) {
    T[] values = enumClass.getEnumConstants();
    if (values != null) {
        for (T enumConst : values) {
            if (enumConst.getId() == id) {
                return enumConst;
            }
        }
    }

    return null;
}
```

This method could be then used like this:

```
MyUtil.getInstance().getById(MyEnum.class, myEnumId);
```

Share Improve this answer Follow

answered May 19, 2010 at 15:42

Marius Burz

4,635 • 2 • 19 • 28