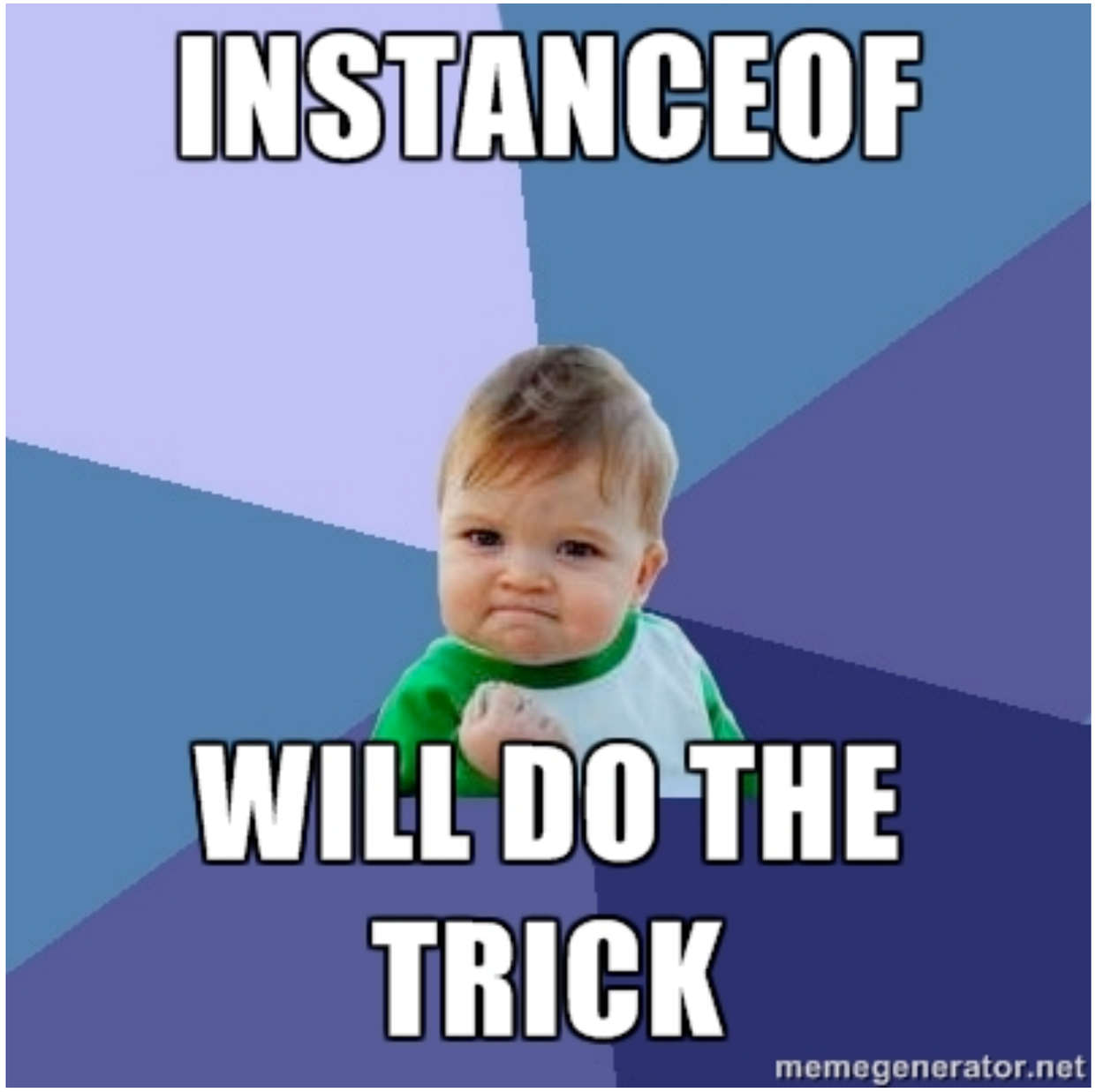
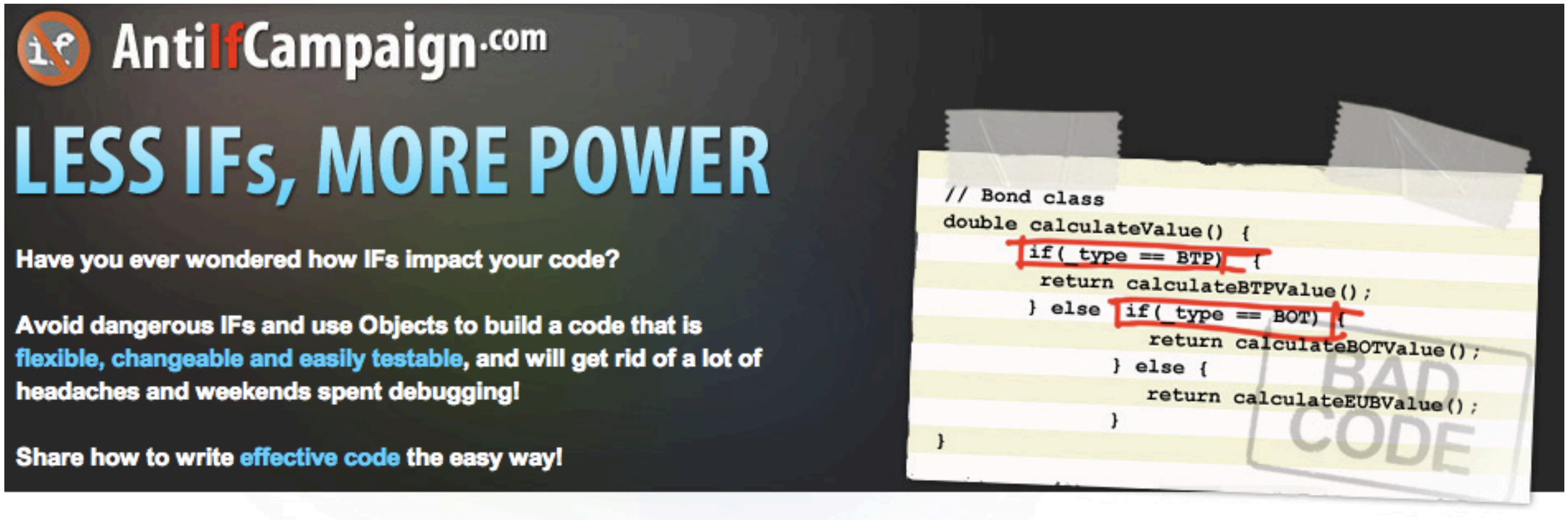
Let’s look at some bad code today

Tijs van der Storm

Things...

- Instanceof
- Meta-level and object-level
- Different things should be different


INSTANCEOF
WILL DO THE TRICK
memegenerator.net



http://www.antiifcampaign.com/

Code smell

- Switch case statements / instanceof usages
- Hand coding dispatch
- We have objects for it
- + get feedback from compiler (Java/C#)
- Be humble, use your brain cpu cycles well!

```java
public final void appendQuestion(Question question) {
    this.registry.addQuestion(question);
    final Type type = question.getType();
    final String name = question.getIdentName();
    String input = new String();
    if (type instanceof BooleanType) {
        input = this.templates.input(name, InputTypes.BOOLEAN);
    }
    if (type instanceof Money) {
        input = this.templates.input(name, InputTypes.MONEY);
    }
    if (type instanceof StrType) {
        input = this.templates.input(name, InputTypes.STRING);
    }
    this.appendToBody(this.templates.question(
         question.getContent().toString(), input));
}
```

```java
@Override
public Value visit(EqualTo astNode, Context param) {
    final Value left = astNode.getLeftExpression()
                              .accept(this, param),
                 right = astNode.getRightExpression()
                              .accept(this, param);

    if(left instanceof Bool && right instanceof Bool)
        return ((Bool)left).isEqualTo((Bool)right);
    else if(left instanceof Int && right instanceof Int)
        return ((Int)left).isEqualTo((Int)right);
    else if(left instanceof Str && right instanceof Str)
        return ((Str)left).isEqualTo((Str)right);
    else
        return new Bool(false);
}
```

```
private JComponent createControlFromType( Type type,
      Value value, boolean editable ) {
    JComponent component = type.accept( this );
    component.setEnabled( editable );

    if ( component instanceof JCheckBox ) {
       ( (JCheckBox) component ).setSelected(
                ( (Boolean) value ).getValue() );
    }
    else if ( component instanceof JTextField ) {
       ( (JTextField) component ).setText(
       value.getValue().toString() );
    }

    return component;
}
```

```
private void findDependencies( LinkedList<Ident> list,
    Expression expression ) {
    if ( expression instanceof BinaryExpression ) {
       this.findDependencies( list,
    ( (BinaryExpression) expression ).getLhs() );
       this.findDependencies( list,
    ( (BinaryExpression) expression ).getRhs() );
    }
    else if ( expression instanceof UnaryExpression ) {
       this.findDependencies( list,
    ( (UnaryExpression) expression ).getExpression() );
    }
    else if ( expression instanceof Ident ) {
       list.add( (Ident) expression );
    }
}
```

```java
@Override
 public PrintResult visit(Type typeDescription) {
    PrintResult pres = null;

    if (typeDescription.getClass() == BooleanType.class) {
       pres = new PrintResult(" boolean ");
    }
    if (typeDescription.getClass() == StringType.class) {
       pres = new PrintResult(" string ");
    }
    if (typeDescription.getClass() == MoneyType.class) {
       pres = new PrintResult(" money ");

    }
    return pres;
 }
```

Classes

- Here classes are used as proxies for the QL types that the classes themselves are representing...

```java
public class Multiply extends BinaryOperation {

  public Multiply(ASTNode leftHandSide, ASTNode rightHandSide) {
    super(leftHandSide, rightHandSide);
  }

   @Override
   public List<Class<?>> getSupportedTypes() {
       List<Class<?>> supportedTypes =
           Arrays.asList(new Class<?>[]{Int.class});
       return Collections.unmodifiableList(supportedTypes);
   }
}
```

Tangling object and meta

- Classes in Java represent Java types not QL types
- QL types are *implemented* in Java, so they can be *objects*
- No need for reflective encoding

```
@Override
 public Boolean visit(Add ast) {
   if (!checkBinary(ast))
     return false;
   Type lhsType = ast.getLhs().typeOf(typeEnv);
   Type rhsType = ast.getRhs().typeOf(typeEnv);
   if (!(lhsType.isCompatibleToNumeric() && rhsType
       .isCompatibleToNumeric())) {
     addError(new Error<Add>(ast, "invalid type for +"));
     return false;
   }
   return true;
 }
```

Parameterized Error type

```java
public class Error<T> extends Type {
   private final T ast;
   private final String str;

   public Error(T ast, String str) {
      this.ast = ast;
      this.str = str;
   }

   public Error(T ast) {
      this.ast = ast;
      this.str = null;
   }

   @Override
   public boolean isCompatibleTo(Type t) {
      return false;
   }
      ....
```

"Error type"

"Message type"

```java
private List<Error<?>> errors;
```

```java
public T getAst() {
    return ast;
 }

 public String getStr() {
    return str;
    }

    @Override
    public <T> T accept(Visitor<T> visitor) {
       return null;
    }
 }
```

When will *T* matter?
What does it give you?

Dead code?

Again: meta level and object level are mixed

These ambiguities, redundances, and deficiencies recall those attributed by Dr. Franz Kuhn to a certain Chinese encyclopedia entitled *Celestial Emporium of Benevolent Knowledge*. On those remote pages it is written that animals are divided into (a) those that belong to the Emperor, (b) embalmed ones, (c) those that are trained, (d) suckling pigs, (e) mermaids, (f) fabulous ones, (g) stray dogs, (h) those that are included in this classification, (i) those that tremble as if they were mad, (j) innumerable ones, (k) those drawn with a very fine camel's hair brush, (l) others, (m) those that have just broken a flower vase, (n) those that resemble flies from a distance.

Jorge Luis Borges

Different types for different things

- Separate hierarchies
  - Types
  - Expressions
  - “Statements” (Questions)
  - Forms
- Java “extends” to be read as “*x* **IS A** *y*”

```
public abstract class Type extends Expr implements Returns {
    public Type(){

    }
}
```

```
public class Question extends Expr implements
     QuestionnaireItemInterface {

   private QuestionType questionType;
   private String questionIdentifier;
   private String questionLabel;
   private Expr valueExpr = null;
   public Question(String qIdentifier, String qLabel,
                   QuestionType qType) {
      this.questionIdentifier = qIdentifier;
      this.questionLabel = qLabel;
      this.questionType = qType;
   }

   public Question(String qIdentifier, String qLabel,
                     QuestionType qType, Expr valExpr) {
      this.questionIdentifier = qIdentifier;
      this.questionLabel = qLabel;
      this.questionType = qType;
      this.valueExpr =  valExpr;
   }

}
```

Is a question an expression?

```java
public class Question extends Expr implements IBinaryNode{
   private final Expr ident;
   private final Expr questionBody;

   public Question(Expr id,Expr questionString, Expr answertype){
      this.ident     = id;
      this.questionBody = new QuestionBody(questionString, answertype);
   }

   @Override
   public Expr getLhs() {
      return this.ident;
   }

   @Override
   public Expr getRhs() {
      return this.questionBody;
   }
}
```

Expressionitis

Lhs and Rhs don't have meaning on Questions?

```
public class NullType extends Expr {
   public NullType() {
   }
   public String getValue() {
      return "";
   }
   @Override
   public boolean isCompatibleTo(Expr t) {
      return false;
   }
   @Override
   public Expr getType(SymbolTable st) {
      return this;
   }

}
```

Defined for all Exprs?

Asking for a type, getting an Expr

Envoy

- You **can** do without instanceof
  - (except maybe in equals())
- Using .class/.getClass makes your code dependent on implementation details
- Smells of object/meta confusion

Envoy ctd

- Use generics when things are generic
- Are you trying to “abuse” the Java type system for your own type system?
- Notice when object/meta-level get confused
- *Rates of change* (Kent Beck)

Envoy ctd

- Think about *intent* of your types
- When are things in the same category?
- Don’t join inheritance hierarchies because for reuse of convenience (only **IS A** counts)