let's **move**
the **java world**

Innovative and Pragmatic Java Source Code Generation

Nikolche Mihajlovski

geecon 2012

Introduction

Hello, world!

```
System.out.println("Hello, GeeCON world!");

Person me = new Person();

me.setFirstName("Nikolche");
me.setLastName("Mihajlovski");
me.setCountry("Macedonia");

System.out.println(me);
```

#define boilerplate

Definition of „boilerplate“:

A copy made with the intention of making other copies from it.

A draft contract that can easily be modified to cover various types of transaction.

A set of instructions incorporated in several places in a computer program.

[Collins English Dictionary]

Definition of „boilerplate code“:

A repetitive code that you don't want to type manually!

[Nikolche Mihajlovski] :)

Imagine...

There are some services:

```
public interface SubtractionService {

        int subtract(int a, int b);

}

public interface MultiplicationService {

        int multiply(int x, int y);

        void doNothing();

}
```

We are going to write a Facade:

```
public class Calculator {

    @Inject
    private SubtractionService subtractionService;

    @Inject
    private MultiplicationService multiplicationService;

    public int subtract(int a, int b) {
        return subtractionService.substract(a, b);
    }

    public int multiply(int x, int y) {
        return multiplicationService.multiply(x, y);
    }

    public void doNothing() {
        multiplicationService.doNothing();
    }

}
```

Let's Be Smart!

There are some services:

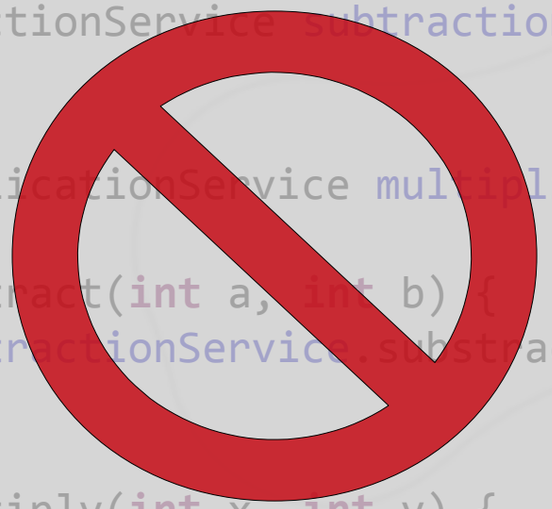
```
public interface SubtractionService {

        int subtract(int a, int b);

}

public interface MultiplicationService {

        int multiply(int x, int y);

        void doNothing();

}
```

Synchronized changes

DRY!

We are going to write a Facade:

```
public class Calculator {

    @Inject
    private SubtractionService subtractionService;

    @Inject
    private MultiplicationService multiplicationService;

    public int subtract(int a, int b) {
        return subtractionService.subtract(a, b);
    }

    public int multiply(int x, int y) {
        return multiplicationService.multiply(x, y);
    }

    public void doNothing() {
        multiplicationService.doNothing();
    }

}
```



Don't write this kind of code manually!

Let's generate it!

Why code generation?

- Productivity boost
- Consistent corectness
- Consistent style
- Consistent design
- DRY principle
- Quick massive changes

Imagine The Ideal Generator...

Increases productivity

- easy to learn
- minimal effort
- easy to debug
- minimal turnaround time
- integrates well

Generates high-quality code

- well-formated code
- well-organized imports

Reusable

- reusable generator
- reusable logic
- reusable code appearance

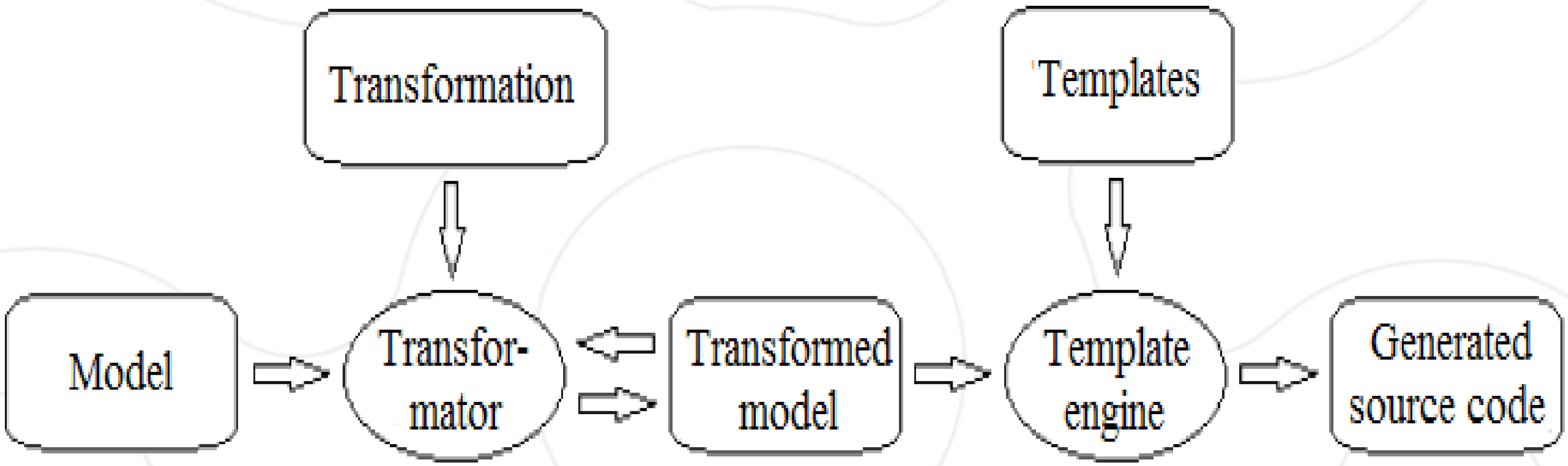
Flexible

- customizations
- work-arounds & hacks

Scalable

- performance
- code quality

Code Generation Workflow



Model-Driven Engineering

Java Model Example

Java source code as model:

```
@BeanModel
public class PersonModel {

    long id;

    String name;

}
```

Code model transformation algorithm:

- *Replace name suffix „Model" with „Gen"*
- *Remove annotations*
- *For each field in the model:*
    - *make the field private*
    - *add getter for the field*
    - *add setter for the field*

Generated Java code as result:

```
public class PersonGen {

    private long id;

    private String name;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

}
```

Java Model Benefits

Pure Java – easy!

- You know Java, right?

Java annotations

- Meta-data collocation (DRY)

Language advantages:

- Type-safe code
- Compile-time check
- IDE support

Which generator?

Many existing code generators:

- Annotated code as model
- Problem-specific
- DAO, Builder, DTO etc.
- Limited capabilities
- Customization?
- They can't serve your **custom** demands!

There is no magical code generator! :(

Write your own code generator!

My own code generator???

Your Own Generator

Java APT (Annotation Processing Tool)

- Part of Java compiler
- Exposes source code model
- Abstract syntax tree
- Read-only view
- Advantages
- Reliable
- Standardized

Shortcomings:

- Minimalistic model
- Low level of abstraction

APT-Based Generator

```
@Override
public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv) {
        if (!roundEnv.processingOver()) {
                for (String annotationType : getSupportedAnnotationTypes()) {
                        Set<? extends Element> annotatedElements = roundEnv.getElementsAnnotatedWith(
                                        processingEnv.getElementUtils().getTypeElement(annotationType));
                        for (Element annotatedElement : annotatedElements) {
                                OutputStream fileStream = null;

                                String elementPackage = processingEnv.getElementUtils()
                                                .getPackageOf(annotatedElement).getQualifiedName().toString();

                                try {
                                        JavaFileObject sourceFile = processingEnv.getFiler().createSourceFile(
                                                        elementPackage + "." + annotatedElement.getSimpleName().toString()
                                                        + getGeneratedFileSuffix(), annotatedElement);
                                        fileStream = sourceFile.openOutputStream();
                                } catch (IOException e) {
                                        processingEnv.getMessager().printMessage(Kind.ERROR,
                                                        "Unable to generate" + " target class for elmenet [reason: "
                                                        + e.toString() + "]", annotatedElement);
                                }

                                PrintWriter pw = new PrintWriter(fileStream);
                                pw.println("package " + elementPackage + ";");
                                pw.println();
                                pw.println("public class " + annotatedElement.getSimpleName() + "Gen"
                                                + getGeneratedFileSuffix() + "{");
                                pw.println("}");
                                pw.flush();
                                pw.close();
                        }
                }
        } else {
                processingEnv.getMessager().printMessage(Kind.NOTE, "Processing finished");
        }
        return false;
}
```

It's Time for a Cake!

2012 geecon

A Piece of Cake

JAnnocessor (code generation == a piece of cake)

- Generic annotation processing framework
  - Not coupled to code generation
- Generic code generation framework
  - Not coupled to annotation processing
- Combination of both
- Annotation-driven code generation framework
- High-level abstraction over APT

Hello, JAnnocessor!

```
public class Processors {

    @Annotated(MyAnnotation.class)
    @Types(JavaInterface.class)
    public CodeProcessor<JavaInterface> generateFacade() {
        return new MyFirstProcessor();
    }

}
```

```
public class MyFirstProcessor implements CodeProcessor<JavaInterface> {

    public void process(PowerList<JavaInterface> elements, ProcessingContext context) {
        for (JavaInterface element : elements) {

            JavaClass clazz = New.class(element.getName() + "Gen");

            JavaPackage pkg = element.getParent();
            pkg.getClasses().add(clazz);

            context.generateCode(clazz, true);
        }
    }

}
```

Back to the Facade!

Remember the services?

```
public interface SubtractionService {

        int subtract(int a, int b);

}

public interface MultiplicationService {

        int multiply(int x, int y);

        void doNothing();

}
```

We are going to ~~write~~ generate the Facade:

```
public class Calculator {

    @Inject
    private SubtractionService subtractionService;

    @Inject
    private MultiplicationService multiplicationService;

    public int subtract(int a, int b) {
        return subtractionService.substract(a, b);
    }

    public int multiply(int x, int y) {
        return multiplicationService.multiply(x, y);
    }

    public void doNothing() {
        multiplicationService.doNothing();
    }

}
```

Generating Facade

```
public class Processors {
	@Annotated(GenerateFacade.class)
	@Types(JavaInterface.class)
	public CodeProcessor<JavaInterface> generateFacade() {
		return new FacadeGenerator();
	}
}

public class FacadeGenerator implements CodeProcessor<JavaInterface> {

	public void process(PowerList<JavaInterface> services, ProcessingContext context) {
		JavaClass facade = New.classs("Calculator");
		New.packagee("example.calculator.facade").getClasses().add(facade);

		for (JavaInterface service : services) {
			String delegateName = service.getName().getUncapitalized();
			JavaField delegate = New.field(service.getType(), delegateName);
			delegate.getMetadata().add(New.metadata(Inject.class));
			facade.getFields().add(delegate);

			for (JavaMethod method : service.getMethods()) {
				facade.getMethods().add(Methods.delegator(method, delegate));
			}
		}

		context.generateCode(facade, true);
	}
}
```

What about Productivity?

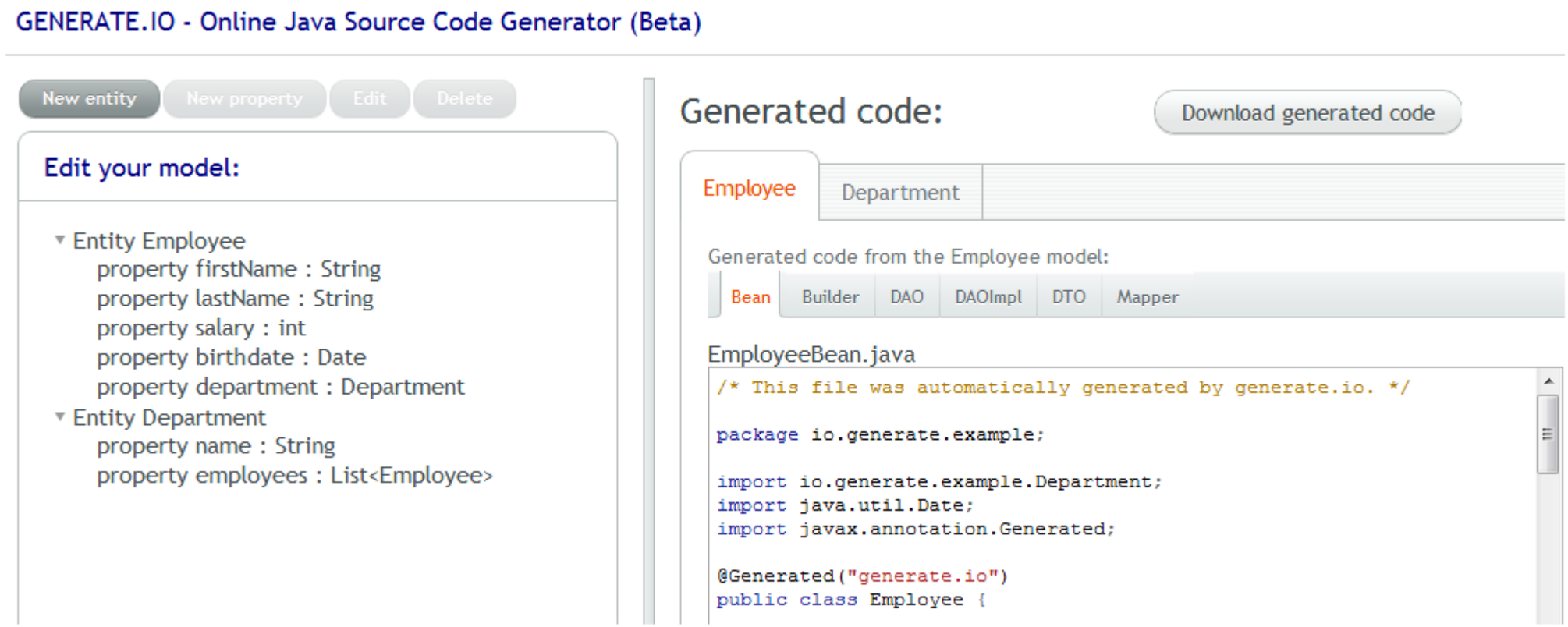
JAnnocessor Features

- Annotation-based configuration
- High-level & rich domain model
- Expressive model transformation
- Graphical UI for real-time debugging
- Hot swap of templates and code processors
- Merge of generated and hand-written code
- Smart imports' organization
- Maven plug-in
- Out-of-the-box commons
  - annotations, processors, templates, merger
- Powerful and convenient collections in domain model
- Integration through Java APT
- Logging bridge from SLF4J to Java APT

For the lazy developers...

GENERATE.IO (Beta)

Simple & Free Code Generation Web Tool



Powered by JAnnocessor!

2012 geecon

Finally { ... }

JAnnocessor is:

- Open-source (GPL)
- ready to be used!
- bootstraped
- in Maven Central (archetype included)

Visit:

- http://jannocessor.googlecode.com
- http://www.jannocessor.org
- http://generate.io

Questions?