

Quality Assurance Generators

Part II - Lecture 10

Generating Code

03/10/2012

SOFTENG 254

The University of Auckland | New Zealand



"...handle an enormous number of variants for different countries and brands"

(<http://www.elektrobit.com>)



R. Engelhardt

"A significant decrease of coding errors due to the extensive use of automatic code generation. For the Airbus A340 project, up to 70% of the code has been automatically generated."

(<http://www.esterel-technologies.com>)



Today's Outline

- Generators
- Generator Type Safety
- Java Emitter Templates (JET)

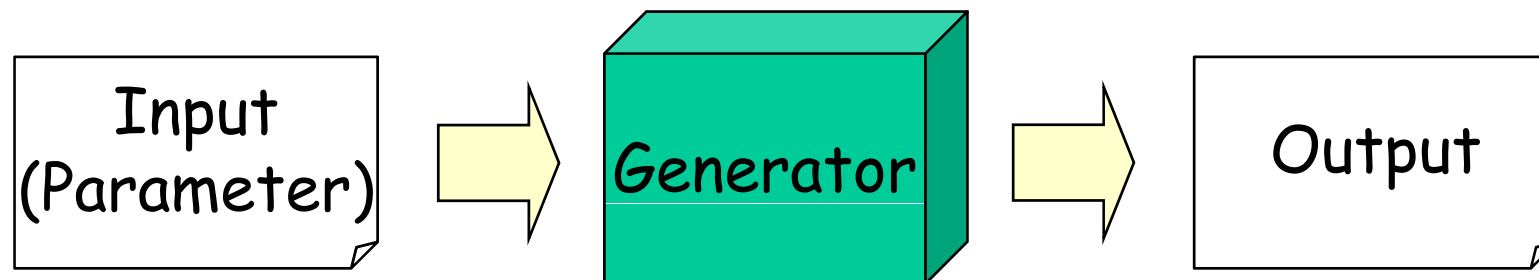
Generators



*"The machine yes, the machine,
never wastes anybody's time,
never watches the foreman,
never talks back"
(Carl Sandburg)*

Generators

- Generators are programs that can **generate certain artefacts**
- They automate the creation of artefacts that have a well-understood, very **regular structure**
- Generated artefacts usually vary with **generator input**
- Examples:
 - Compiler: generates binary code from source code
 - JavaDoc: generates HTML from source code comments
 - Some UML tools: generating source code from class diagram
 - Servlets: generating HTML pages
 - Java Server Pages (JSP): generating Servlets



Generative Programming

- There are programming “**routine tasks**” that are always similar
- Only slight variation depending on some **parameters**
- Generative programming tries to automate these tasks with **parameterized program generators** for different kinds of program parts
- Program generators are **meta-programs**: programs that deal with other programs or themselves
- Meta-programming can be sophisticated and potentially **unsafe**
- Different approaches to generative programming:
 - **External tools**: stand-alone programs that usually perform a particular program generation task (e.g. compiler, compiler-generator)
 - **Generative language features**: constructs for generation integrated into a programming language
Usually transformation of new high-level constructs in lower-level ones (often simply called ‘**macros**’)

Common Patterns & Applications

- **Class extensions**

Input: class to be extended & additional information

Output: subclass with additional functionality

E.g. clone, hashCode, equals, print, serialize, copy

- **Proxies**

Input: type and methods/fields to be hidden

Output: subclass with modified semantics

E.g. monitoring, remoting, resource management, access control

- **Wrappers**

Input: types, methods, fields to be wrapped

Output: wrapper class with appropriate interface

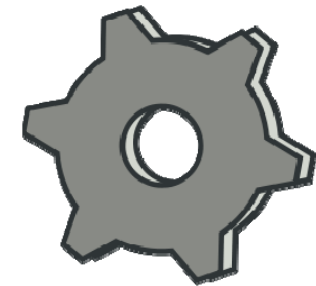
E.g. integration of legacy components

- **Interfaces**

Input: interface description

Output: different kinds of interfaces

E.g. DB interface, GUI, web interface, API



Templates

- Code that generates code can be **confusing**
- Example: constructing an Abstract Syntax Tree (AST) for the generated code

```
JavaClass c = new JavaClass("GeneratedClass");  
c.extends = x;  
c.fields.add(new Field(TYPE_INTEGER, "myVar"));
```

- Hard to see what is actually created

- Better approach: **templates**

```
class GeneratedClass extends @x@ {  
    Integer myVar;  
}
```

- Output is given in its **natural form**
- At places where output varies, we insert **generator code** (e.g. @x@)

Example: Getter&Setter Generator

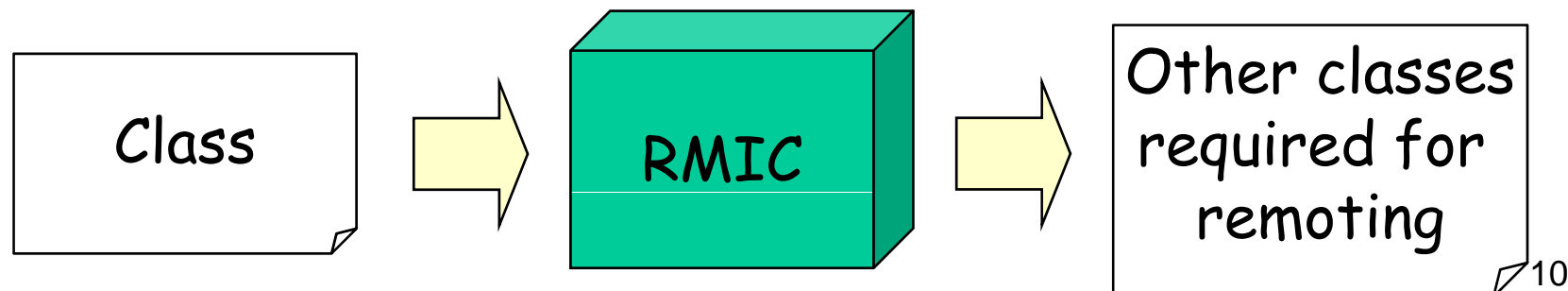
- Java convention: classes provide getter- and setter-methods for access of member variables
- Can be useful, e.g. for observer pattern
- Simple getters and setters are purely routine work
- We can automate it with a generator:

```
class Person {  
    String name;  
    int age;  
}
```

```
class PersonWithGetterSetter {  
    String name;  
    int age;  
    String getName() { return name; }  
    void setName(String v) {  
        name = v;  
    }  
    int getAge() { return age; }  
    void setAge(int v) { age = v; }  
}
```

Example: Remote Method Calls

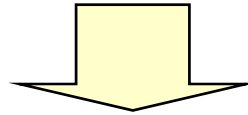
- Call methods of objects on other computers as if they were local
- Requires new class: **client stub**
 - For object **representing the remote object** locally
 - Has **same signature** as remote object (i.e. same interface)
 - But different method implementation:
 1. Send method call request and parameters to server
 2. Wait and receive method return value
- In Java: client stubs and other classes are generated by external generator tool RMIC (Remote Method Invocation Compiler)



Stub Generator

Example: remote matrix multiplication

```
class Matrix implements MatrixInterface {  
    ...  
    Matrix multiply(Matrix m) { ... }  
}
```



Pseudo-code:

```
class MatrixStub implements MatrixInterface {  
    Url remoteObject;  
    ...  
    Matrix multiply(Matrix m) {  
        send(remoteObject, REQUEST_FOR_MULTIPLY, m);  
        return (Matrix) receive(remoteObject);  
    }  
}
```

Aims of Code Generation

1. More efficient **development**
 - Adaptability and reuse
 - Control complexity
 - Clearer structure (e.g. templates)
 - Better handling of multiple variants (e.g. parameterization)
2. Avoid development **mistakes** by reducing human involvement where it is unnecessary
3. More efficient **usage** through adaptability and adaptivity (e.g. dynamic reflection)
4. **Performance** gain at runtime through adapted components (e.g. generation of optimized code)

Generator Type Safety

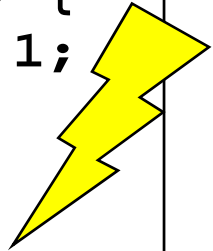


*Safety doesn't happen
by accident.*

Generator Type Safety

- Type systems can detect potential **execution errors** statically

```
int m(String s) {  
    int y = s + 1;  
    m(y, 3);  
    return s;  
}
```



- Generators are meta-programs with new sources of execution errors: **generator type errors**
 - parts of the generator program that can potentially generate malformed code
 - which in turn may cause execution errors
- Need new kind of type system for detecting parts in generators that can potentially generate ill-typed code (**generator type system**)
- **Generator type safety**: property of a generator not to be able to generate ill-typed code
- Unfortunately generator type safety is usually not guaranteed...

Generator Type Errors

- By type-checking generator output we may detect generator type errors

```
class C(String ID) {  
    String @ID@ = 1;  
}
```

Always generates ill-typed code

- But some generator type errors only produce ill-typed code for some parameters, not for others

```
class C(String ID) {  
    int x;  
    int @ID@;  
}
```

Works fine for most IDs
but not for "x"
(lexical collision)

- This makes it more difficult to find generator type errors

Generator Type Errors

```
class C(String ID) {  
    void m() {  
        int @ID@ = 1;  
        x++;  
    }  
}
```

1. Output only correct
iff ID equals "x"

```
class C(Class T) {  
    @T@ x = new Button();  
}
```

2. Output only correct
iff T supertype of
Button

```
class C(String X) {  
    @if(X.Equals("hello")) {  
        String y = "world";  
    }  
    void m() {  
        Console.WriteLine(y);  
    }  
}
```

3. Output only correct
iff x equals "hello"

Generator Type Errors

```
class C(Type T) {  
    @T@ x = 1;  
}
```

```
class C(Type S, Type T) {  
    @foreach(F in S.GetFields()) {  
        @F.FieldType@ @F.FieldName@;  
    }  
    void m() {  
        @foreach(F in T.GetFields())  
        {  
            Console.WriteLine(  
                this.@F.FieldName@);  
        }  
    }  
}
```

1. Only correct iff
1 element of
type T

2. Only correct iff
T's field names
are subset of
s's field names

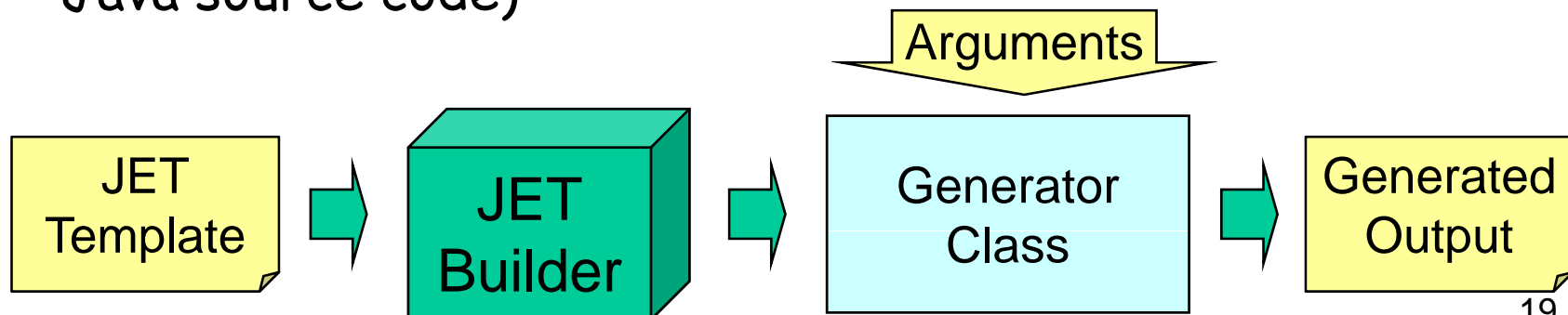
E.g. if $T=S$

Java Emitter Templates (JET)



Java Emitter Templates (JET)

- Generator technology based on templates
- Part of the Eclipse Modeling Framework (EMF)
- JSP-like syntax (actually a subset of JSP)
- Idea:
 1. Developer creates parameterized **templates** (text files that end with jet)
 2. Each template is transformed into a **generator class** (template implementation class)
 3. Generator classes can be used to **generate** something, e.g. source code
- Can be used to generate any kind of text file (not just Java source code)



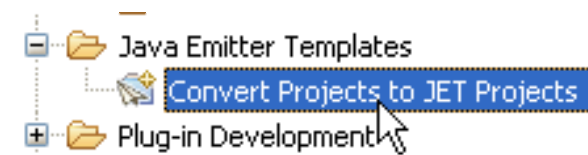
Using JET in Eclipse

Ensure EMF is installed (find it under Modeling); use update site for your Eclipse version, e.g.

<http://download.eclipse.org/releases/juno>

1. Create a new Java project
2. Convert the Project to a JET Project

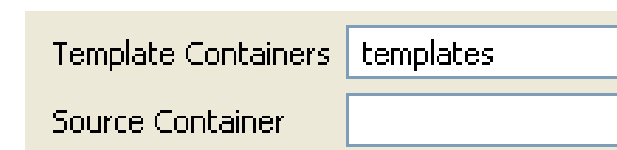
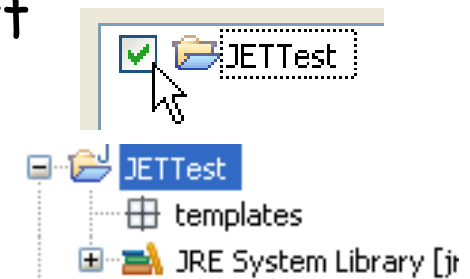
- Right-click it in the package explorer; *New -> Other...*
- Select "Convert Projects to JET..."; Next
- Select your project; Finish



Now the project has a "templates" folder

3. To configure JET:

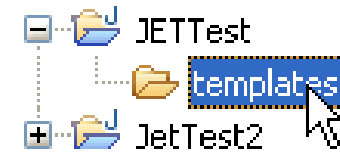
- Right-click project in package explorer; Properties
- Select "JET Settings"
- Choose folders for templates and Java source code



"Hello world" Generator

Create a JET Template File

1. From the menu choose *File -> New -> File*
2. Select the templates directory as parent folder; call the file `helloworld.txtjet`
3. After OK error message pops up: "The jet directive is missing..."; this is normal, just close it
4. Edit `helloworld.txtjet`



```
<%@ jet package="hello" class="HelloWorldTemplate" %>
Hello, world!
```

As soon as you save, package hello with template implementation class HelloWorldTemplate is generated



Convention: suffix of template is suffix of output + "jet"

E.g. `.txt` -> `.txtjet`, `.java` -> `.javajet`, `.xml` -> `.xmljet`

Using the "Hello world" Generator

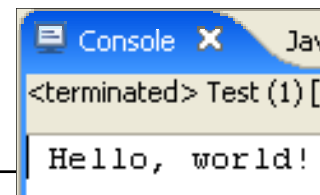
Excerpt from HelloWorldTemplate:

```
protected final String TEXT_1 = NL + "Hello, world!";

public String generate(Object argument)
{
    final StringBuffer stringBuffer = new StringBuffer();
    stringBuffer.append(TEXT_1);
    return stringBuffer.toString();
}
```

Create a new class in package hello that uses it:

```
public class Test {
    public static void main(String[] args) {
        HelloWorldTemplate t = new HelloWorldTemplate();
        String result = t.generate(null);
        System.out.println(result);
    }
}
```





Today's Summary

- **Generators** are programs that can generate certain artefacts
- **Generative programming** tries to generate program parts, e.g. class extensions, proxies, wrappers, interfaces
- With **templates** generator output can be given in its natural form
- **Generator type errors** are parts of the generator program that can potentially generate malformed code
- **Java Emitter Templates (JET)** are a popular generator technology for Eclipse

Reference:

Eclipse JET Tutorial. http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html

Quiz

1. What can generators be used for? Name five examples.
2. What is a generator type error? Give a definition.
3. Can you give pseudo-code examples for three different generator type errors?