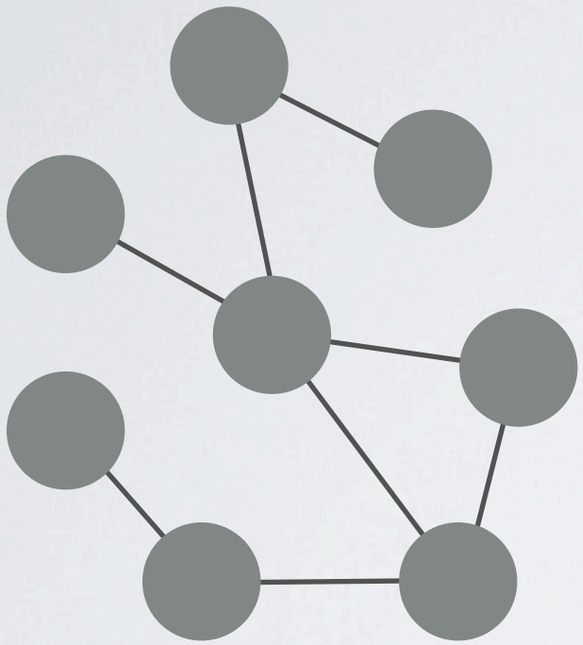
A DSL for Generating Code

A DSL for Generating Code

A DSL for Generating Code

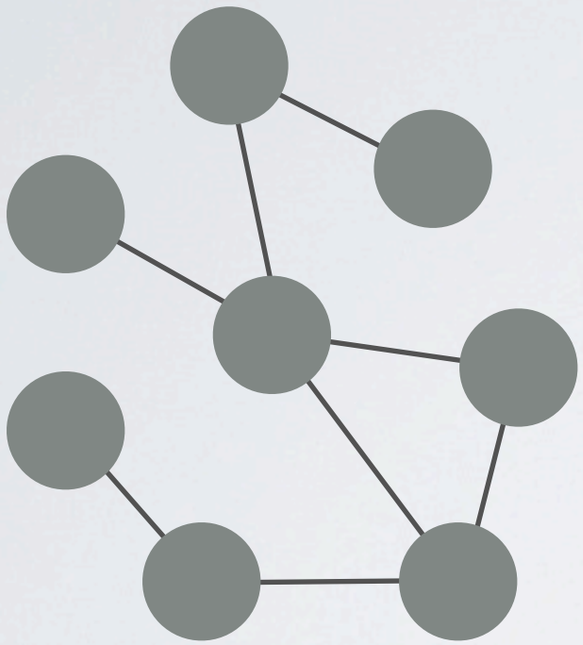
Model

Code



Code Generator

Model



Code Generator

Code

```
package base;


public class Address {
	private String street;

	public String getStreet() {
		return street;
	}

	public void setStreet(String street) {
		this.street = street;
	}

	private String city;

	public String getCity() {
		return city;
	}

	public void setCity(String city) {
		this.city = city;
	}

	private String state;

	public String getState() {
		return state;
	}

	public void setState(String state) {
		this.state = state;
	}

	private String postalCode;

	public String getPostalCode() {
		return postalCode;
	}

	public void setPostalCode(String postalCode) {
		this.postalCode = postalCode;
	}

}
```

BUT I CAN

DO THAT IN Java

Key Challenges in Writing Code Generators

```
package example.codegen;

import org.eclipse.xtext.common.types.JvmFormalParameter;
import org.eclipse.xtext.common.types.JvmTypeReference;
import org.eclipse.xtext.example.domainmodel.domainmodel.Entity;
import org.eclipse.xtext.example.domainmodel.domainmodel.Feature;
import org.eclipse.xtext.example.domainmodel.domainmodel.Operation;
import org.eclipse.xtext.example.domainmodel.domainmodel.Property;

public class JavaGenerator {

    public String compile(Entity e) {
        StringBuilder b = new StringBuilder();
        b.append("package " + packageName(e) + "\n");
        b.append("\n");
        b.append("/**\n");
        b.append(" * Automatically generated. Direct modification is futile.\n");
        b.append(" */\n");
        b.append("public class ").append(e.getName());
        if(e.getSuperType() != null) {
            b.append(" extends ").append(compile(e.getSuperType()));
        }
        b.append(" {\n");
        for(Feature f: e.getFeatures()) {
            if(f instanceof Property)
                feature((Property)f);
            else if(f instanceof Operation)
                feature((Operation)f);
        }
        b.append("}\n");
        return b.toString();
```

```
package example.codegen;

import org.eclipse.xtext.common.types.JvmFormalParameter;
import org.eclipse.xtext.common.types.JvmTypeReference;
import org.eclipse.xtext.example.domainmodel.domainmodel.Entity;
import org.eclipse.xtext.example.domainmodel.domainmodel.Feature;
import org.eclipse.xtext.example.domainmodel.domainmodel.Operation;
import org.eclipse.xtext.example.domainmodel.domainmodel.Property;

public class JavaGenerator {

    public String compile(Entity e) {
        StringBuilder b = new StringBuilder();
        b.append("package " + packageName(e) + "\n");
        b.append("\n");
        b.append("/**\n");
        b.append(" * Automatically generated. Direct modification is futile.\n");
        b.append(" */\n");
        b.append("public class ").append(e.getName());
        if(e.getSuperType() != null) {
            b.append(" extends ").append(compile(e.getSuperType()));
        }
        b.append(" {\n");
        for(Feature f: e.getFeatures()) {
            if(f instanceof Property)
                feature((Property)f);
            else if(f instanceof Operation)
                feature((Operation)f);
        }
        b.append("}\n");
        return b.toString();
```

No Multiline String Literals

```
public String compile(Entity e) {
    StringBuilder b = new StringBuilder();
    b.append("package " + packageName(e) + "\n");
    b.append("\n");
    b.append("/**\n");
    b.append(" * Automatically generated. Direct modification is futile.\n");
    b.append(" */\n");
    b.append("public class ").append(e.getName());
    if(e.getSuperType() != null) {
        b.append(" extends ").append(compile(e.getSuperType()));
    }
    b.append(" {\n");
    for(Feature f: e.getFeatures()) {
        if(f instanceof Property)
            feature((Property)f);
        else if(f instanceof Operation)
            feature((Operation)f);
    }
    b.append("}\n");
    return b.toString();
}

public String feature(Property property) {
    StringBuilder b = new StringBuilder();
    String type = compile(property.getType());
    String name = property.getName();
    b.append("// property ").append(name).append("\n");
    b.append("private ").append(type).append(" ").append(name).append("\n");
    b.append("\n");
    b.append("public ").append(type).append("get").append(name).append("() {\n");
    b.append("  return ").append(name).append(";\n");
```

Instanceof Cascades

```
        }
        b.append("}\n");
        return b.toString();
    }

    public String feature(Property property) {
        StringBuilder b = new StringBuilder();
        String type = compile(property.getType());
        String name = property.getName();
        b.append("// property ").append(name).append("\n");
        b.append("private ").append(type).append(" ").append(name).append("\n");
        b.append("\n");
        b.append("public ").append(type).append("get").append(name).append("() {\n");
        b.append("  return ").append(name).append(";\n");
        b.append("}\n");
        b.append("\n");
        b.append("public ").append("set").append(name)
            .append("(").append(type).append(" ").append(name).append("){\n");
        b.append("  this.").append(name).append(" = ").append(name).append(";\n");
        b.append("}\n");
        b.append("\n");
        return b.toString();
    }

    public String feature(Operation operation) {
        StringBuilder b = new StringBuilder();
        String name = operation.getName();
        b.append("// operation ").append(name).append("\n");
        b.append(compile(operation.getType())).append(" ").append(name).append("(");
        boolean isFirst = true;
        for(JvmFormalParameter p: operation.getParams()) {
```

Noisy String Concatenation


Xtend

STATICALLY TYPED

« RICH STRINGS »

CONCISE EXPRESSIONS

POLYMORPHIC

dispatch

COMPILES TO

Java

DEPENDENCY @InjectION


Xtext
TOOLING