Static meta-programming

Václav Pech

NPRG014 2023/2024

http://www.vaclavpech.eu

@vaclav_pech

Last time agenda

Dynamic method call dispatch
Dynamic object creation
Builders
Traits
Categories
Dynamic DSLs

Part 5
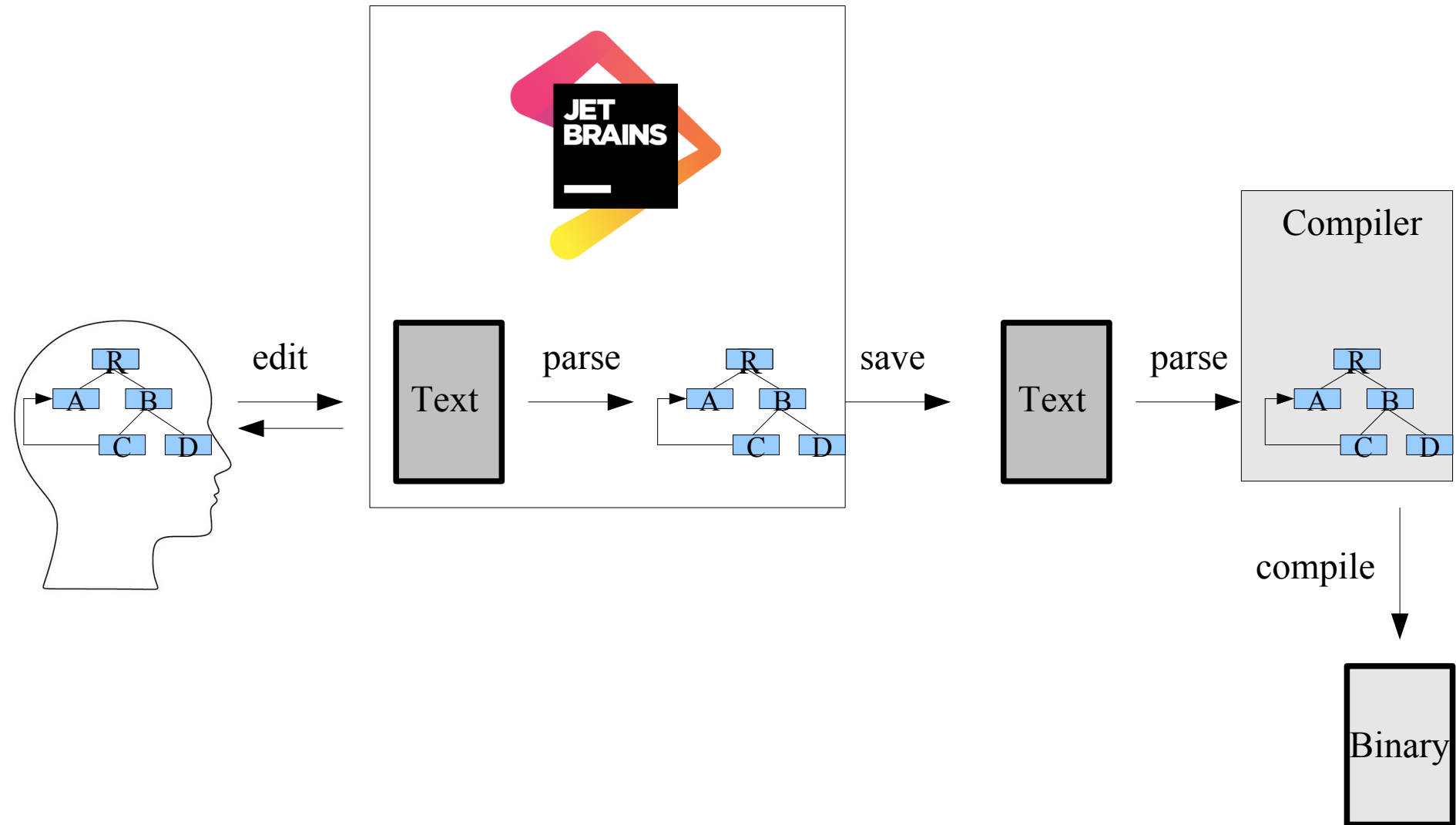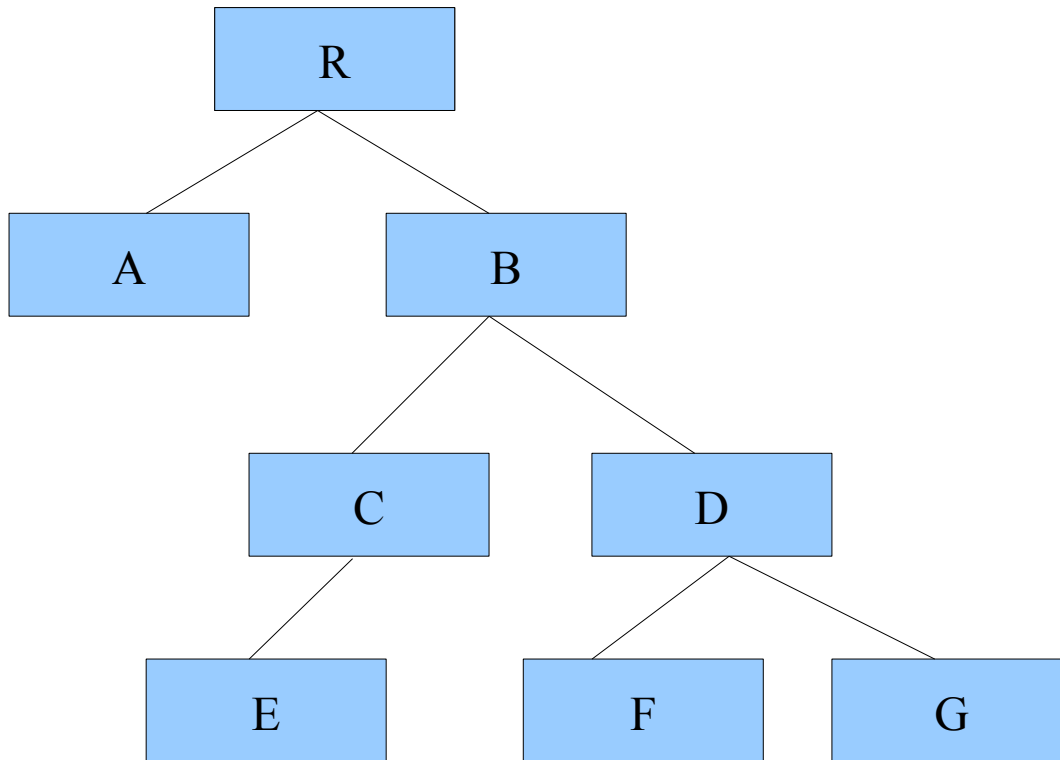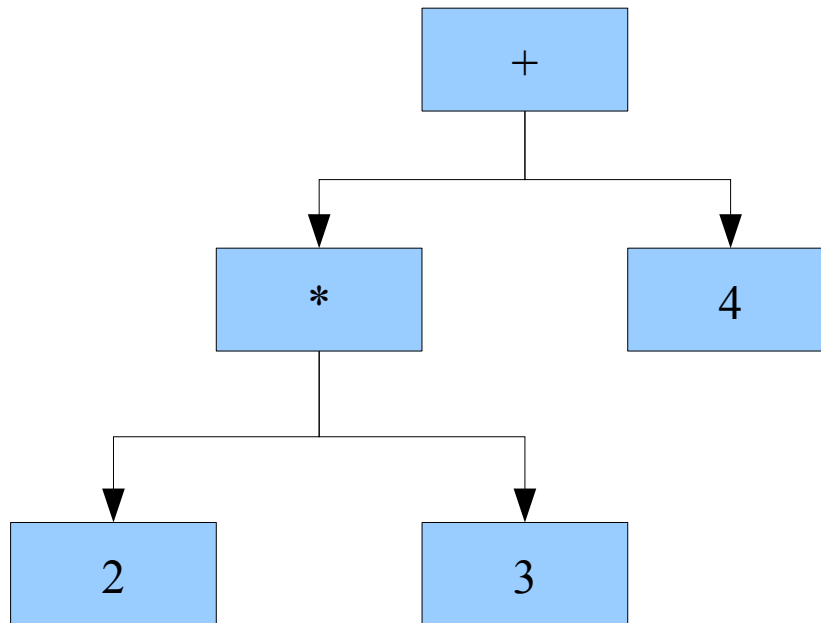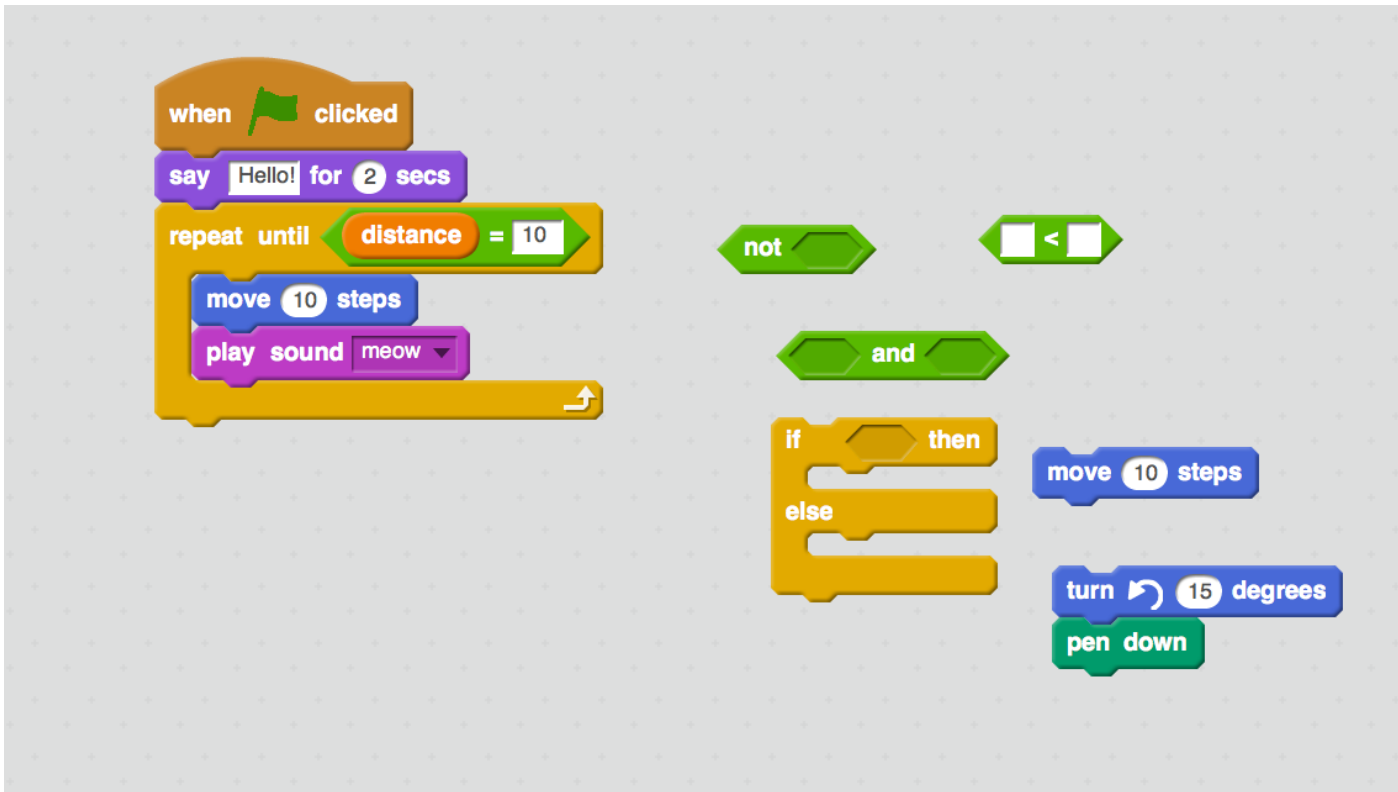
Static meta-programming

AST

```
ClassNode - Calculator
  Constructors
  Methods
    MethodNode - divide10By
    MethodNode - subtract
      Parameter - a
      Parameter - b
      BlockStatement - (2)
        BlockStatement - (1)
          IfStatement
            Boolean - (a > b)
              Not - (a > b)
                Binary - (a > b)
            BlockStatement - (1)
            EmptyStatement
        ReturnStatement - return (a - b)
          Binary - (a - b)
            Variable - a : java.lang.Object
              Parameter - a
            Variable - b : java.lang.Object
              Parameter - b
    MethodNode - this$dist$invoke$1
```

```java
public int subtract(java.lang.Object a, java.lang.Object b) {
    if (!( a > b )) {
        throw new java.lang.Exception('Precondition violated: {a > b}')
    }
    a - b
}
```

Programming

Programs are trees

Lisp (1958)

(+ (* 2 3) 4)

when 🚩 clicked

say `Hello!` for `2` secs

repeat until `distance` = `10`

  move `10` steps

  play sound `meow ▾`

not

`☐` < `☐`

and

if `⬡` then

else

move `10` steps

turn ↺ `15` degrees

pen down

```java
public class Demo {

    private static void foo() {
        System.out.println("Foo called");
    }

    public static void main(string[] args) {

        System.out.println("Application started");

        if (args.length > 0) {
            System.out.println("Supplied arguments");
            for (string value : args) {
                System.out.println("Argument: " + value);
            }
        } else {
            System.out.println("No arguments provided");
        }

        foo();
        System.out.println("Application completed");
    }

}
```
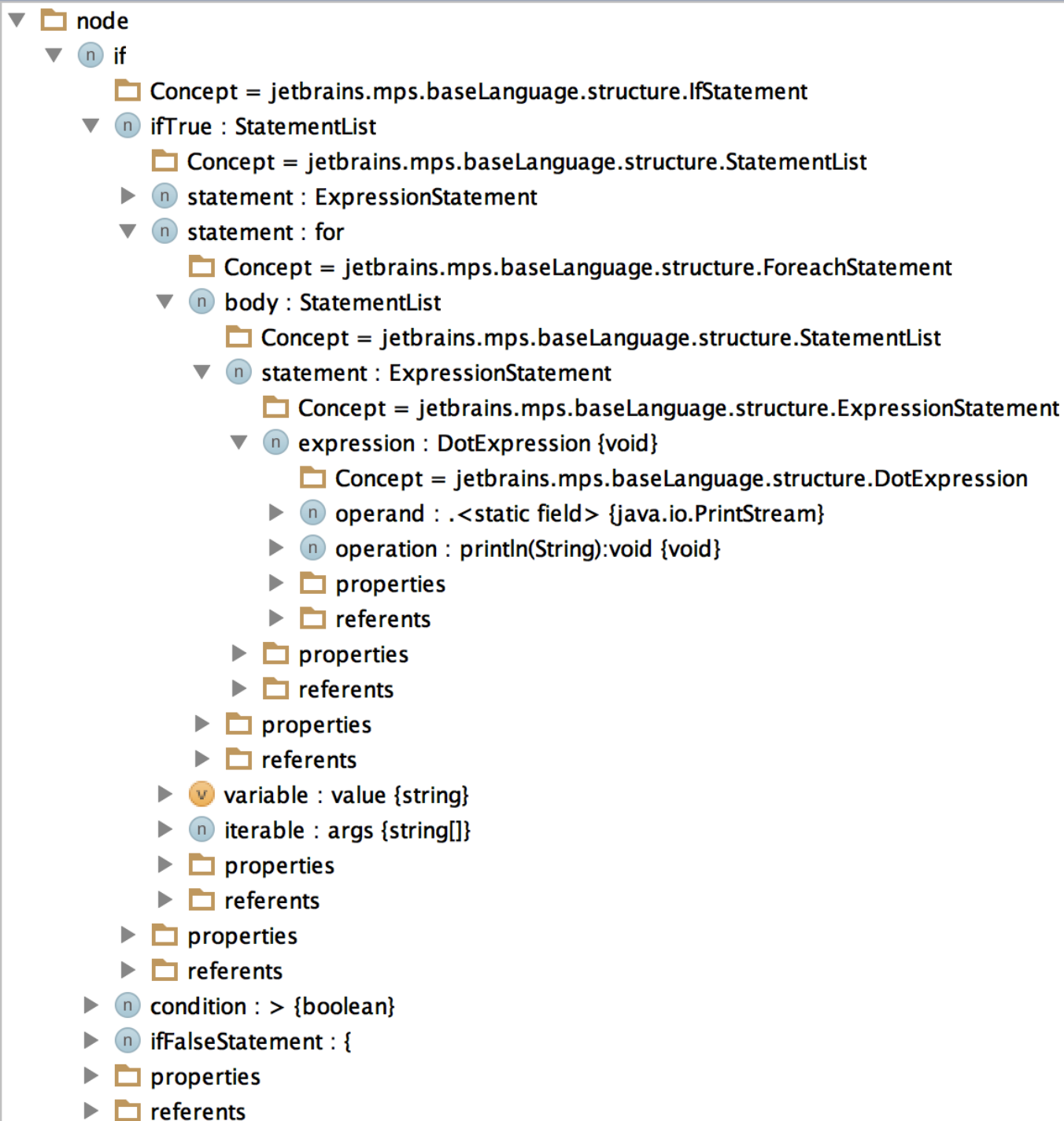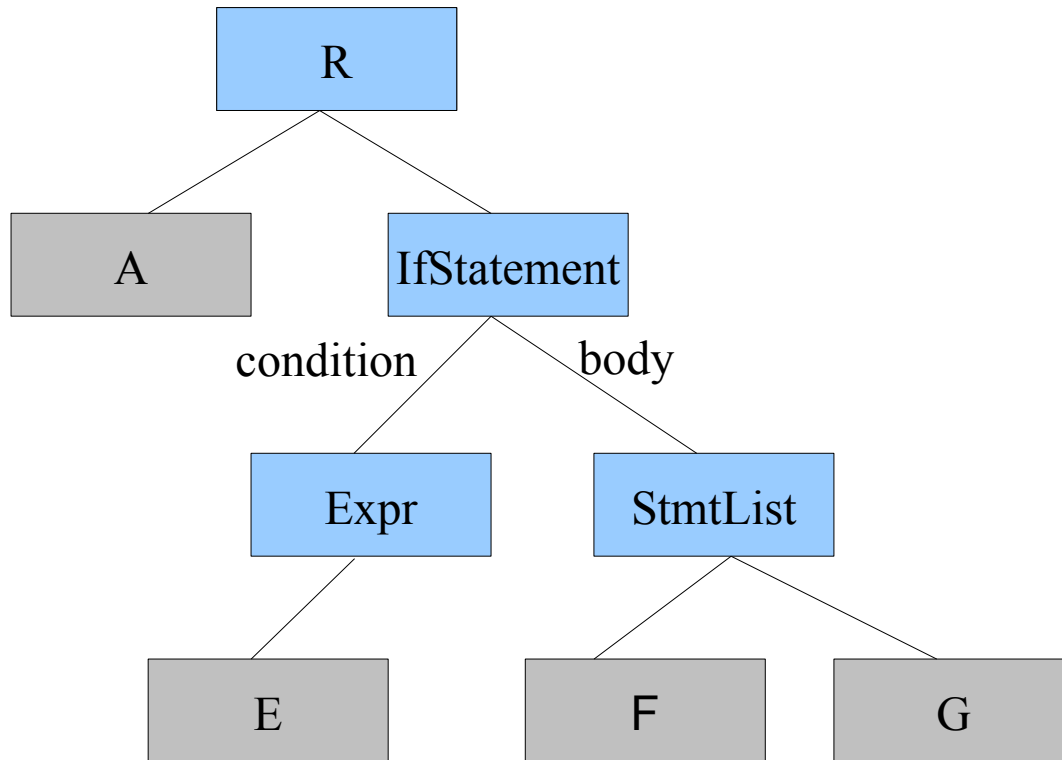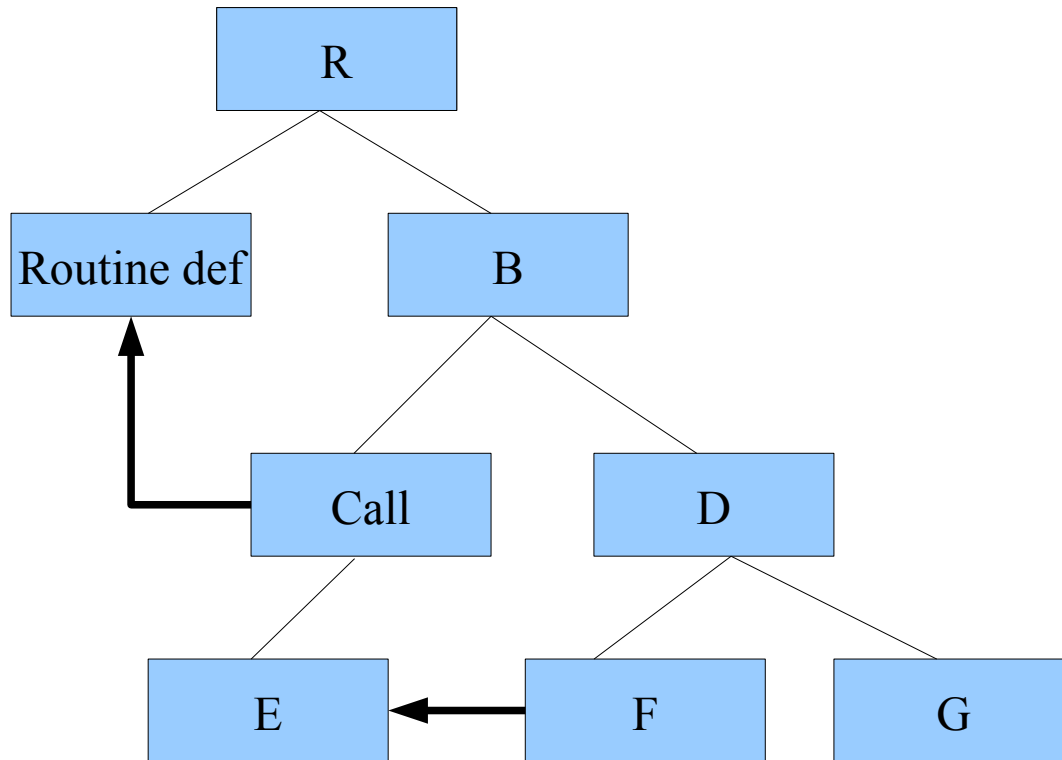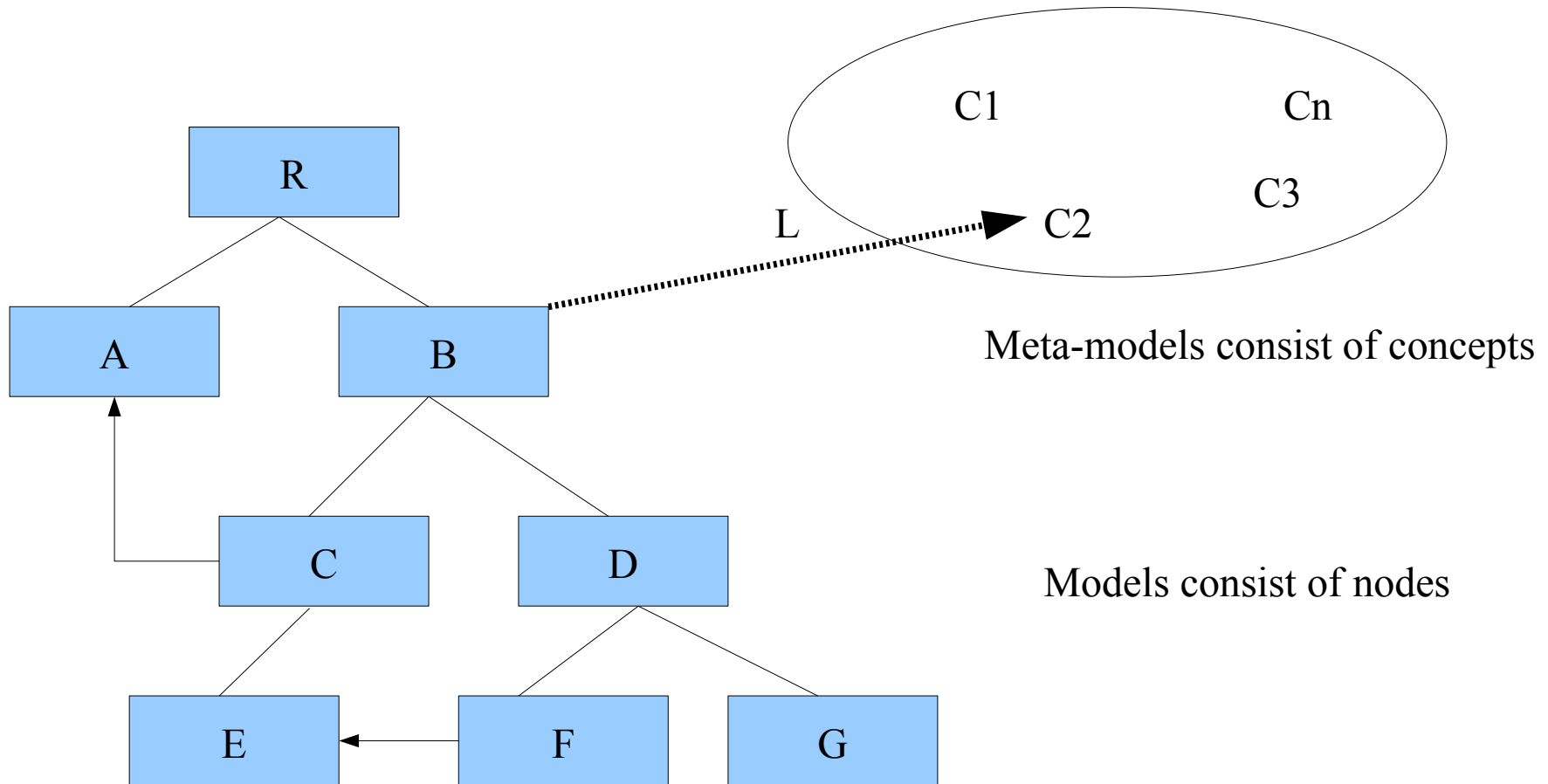
- ▼ 📁 node
  - ▼ ⓝ if
    - 📁 Concept = jetbrains.mps.baseLanguage.structure.IfStatement
    - ▼ ⓝ ifTrue : StatementList
      - 📁 Concept = jetbrains.mps.baseLanguage.structure.StatementList
      - ▶ ⓝ statement : ExpressionStatement
      - ▼ ⓝ statement : for
        - 📁 Concept = jetbrains.mps.baseLanguage.structure.ForeachStatement
        - ▼ ⓝ body : StatementList
          - 📁 Concept = jetbrains.mps.baseLanguage.structure.StatementList
          - ▼ ⓝ statement : ExpressionStatement
            - 📁 Concept = jetbrains.mps.baseLanguage.structure.ExpressionStatement
            - ▼ ⓝ expression : DotExpression {void}
              - 📁 Concept = jetbrains.mps.baseLanguage.structure.DotExpression
              - ▶ ⓝ operand : .<static field> {java.io.PrintStream}
              - ▶ ⓝ operation : println(String):void {void}
              - ▶ 📁 properties
              - ▶ 📁 referents
            - ▶ 📁 properties
            - ▶ 📁 referents
        - ▶ 📁 properties
        - ▶ 📁 referents
      - ▶ ⓥ variable : value {string}
      - ▶ ⓝ iterable : args {string[]}
      - ▶ 📁 properties
      - ▶ 📁 referents
    - ▶ 📁 properties
    - ▶ 📁 referents
  - ▶ ⓝ condition : > {boolean}
  - ▶ ⓝ ifFalseStatement : {
  - ▶ 📁 properties
  - ▶ 📁 referents

Children have roles

References cross the tree

Programs and Languages

Meta-models consist of concepts

Models consist of nodes

Languages are sets of concepts

C1   Cn

C3

L1   C2

L2
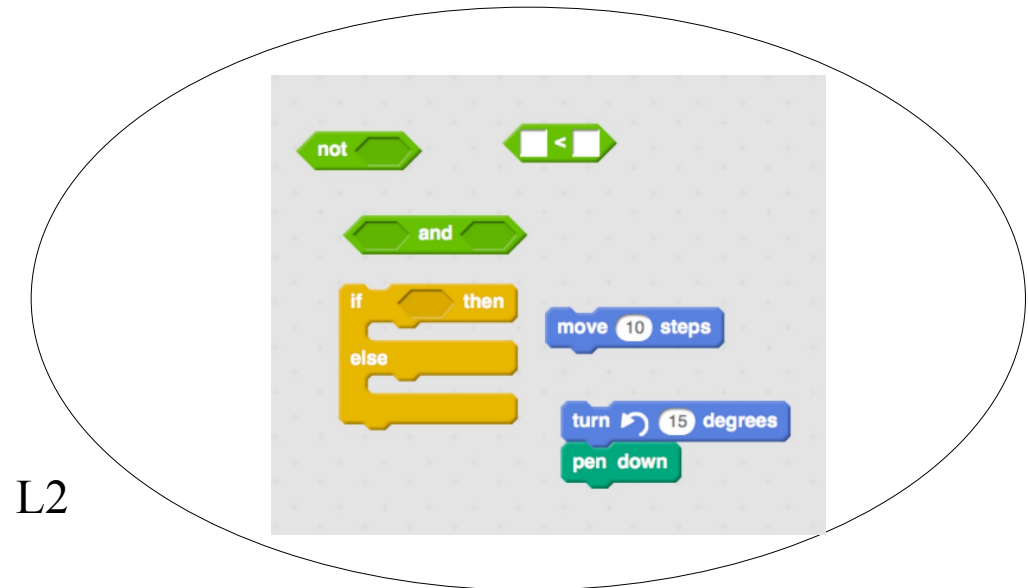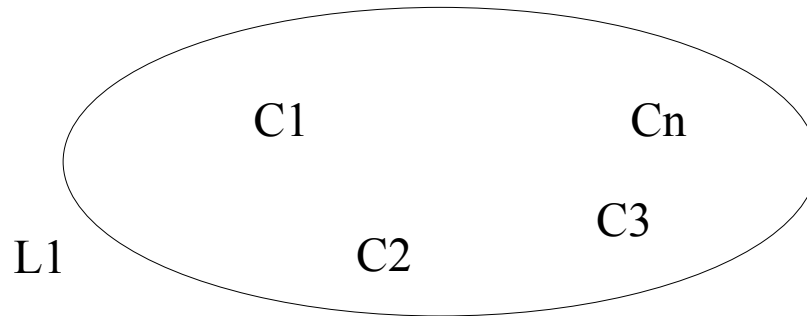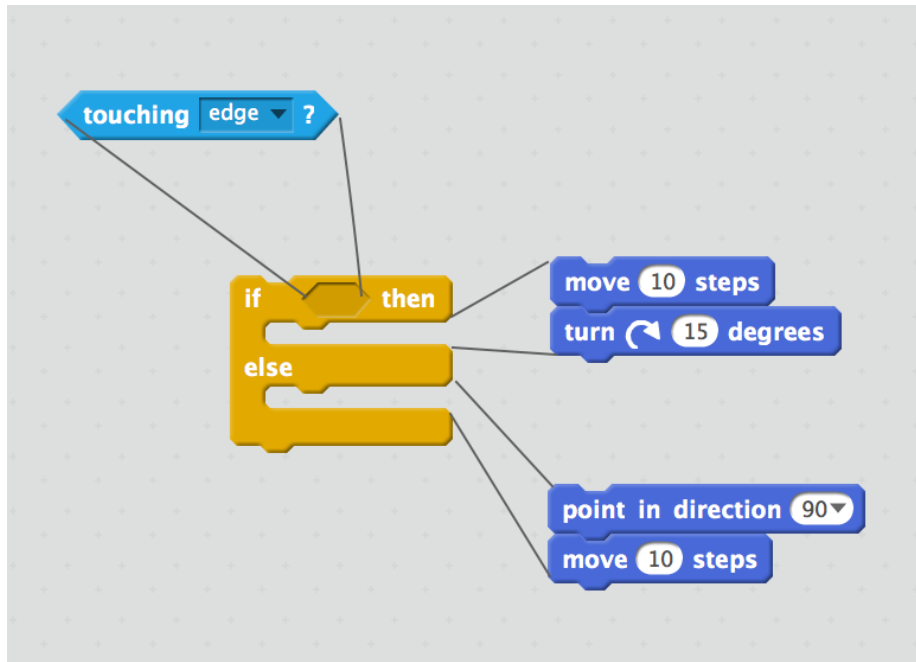
```
...us.groovy.ast.stmt;

/**
 * Represents an if (condition) { ... } else { ... } statement in Groovy
 *
 * @author <a href="mailto:james@coredevelopers.net">James Strachan</a>
 * @version $Revision$
 */
public class IfStatement extends Statement {

    private BooleanExpression booleanExpression;
    private Statement ifBlock;
    private Statement elseBlock;


    public IfStatement(BooleanExpression booleanExpression, Statement ifBlock,
        this.booleanExpression = booleanExpression;
        this.ifBlock = ifBlock;
        this.elseBlock = elseBlock;
    }
```

The Node

```java
public class ASTNode {

    private int lineNumber = -1;
    private int columnNumber = -1;
    private int lastLineNumber = -1;
    private int lastColumnNumber = -1;
    private ListHashMap metaDataMap = null;

    public void visit(GroovyCodeVisitor visitor) {
        throw new RuntimeException("No visit() method implemented for class: "
    }
```

```java
public class TernaryExpression extends Expression {

    private BooleanExpression booleanExpression;
    private Expression trueExpression;
    private Expression falseExpression;
```

```java
public class ElvisOperatorExpression extends TernaryExpression {

    public ElvisOperatorExpression(Expression base, Expression falseExpression) {
        super(getBool(base), base, falseExpression);
    }
```

```java
public class ForStatement extends Statement implements LoopingStatement {
    public static final Parameter FOR_LOOP_DUMMY = new Parameter(ClassHelp

    private Parameter variable;
    private Expression collectionExpression;
    private Statement loopBlock;
    private VariableScope scope;
```

```java
public class MethodNode extends AnnotatedNode implements Opcodes {

    public static final String SCRIPT_BODY_METHOD_KEY = "org.codeha
    private final String name;
    private int modifiers;
    private boolean syntheticPublic;
    private ClassNode returnType;
    private Parameter[] parameters;
    private boolean hasDefaultValue = false;
    private Statement code;
    private boolean dynamicReturnType;
    private VariableScope variableScope;
    private final ClassNode[] exceptions;
    private final boolean staticConstructor;

    // type spec for generics
    private GenericsType[] genericsTypes = null;
    private boolean hasDefault;

    // cached data
    String typeDescriptor;
```

Visitor pattern

Separate class hierarchies from their processing

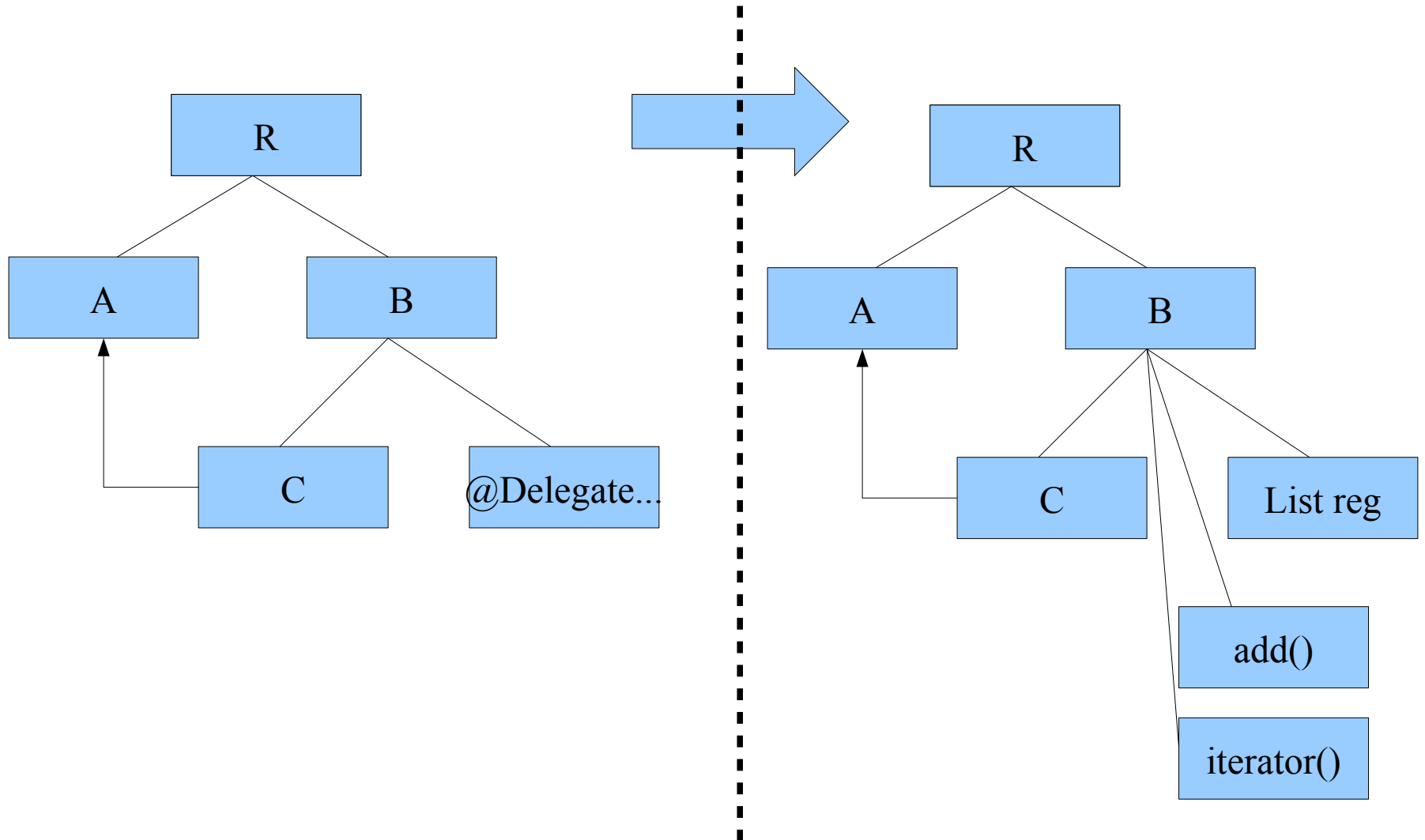# There's no life without trees

- Extending IDEs

- External DSLs

- Internal DSLs in modern languages

  - Groovy AST transformations

  - Scala macros

  - Lisp, Clojure macros

  - Lombok

AST Transformations

```
class Registrations {
    @Delegate List items = []
}

def people = new Registrations()
people.addAll(["Joe", "Dave"])
assert ["Dave", "Joe"] == people.reverse()
```

Ast transformation

@Delegate, @Immutable, @Singleton

@Lazy

@TupleConstructor

@InheritConstructors

@Canonical

@ToString

@EqualsAndHashCode

@Log, @Log4j, @Commons

@Synchronized

@WithReadLock

@WithWriteLock

@AutoClone, @AutoExternalize

...

Type-checking/Static

@TypeChecked, @CompileStatic

```
@TypeChecked
String test(Object val) {
    if (val instanceof String) {
        val.toUpperCase()
    } else if (val instanceof Number) {
        val.intValue() * 2
    }
}
```

Creating AST Transformations

```groovy
@Retention(RetentionPolicy.SOURCE)
@Target([ElementType.METHOD])
@GroovyASTTransformationClass("RequiresTransformation")
public @interface Requires {
    String value() default "true";
}

@GroovyASTTransformation(phase = SEMANTIC_ANALYSIS)
public class RequiresTransformation implements ASTTransformation {

    public void visit(ASTNode[] astNodes, SourceUnit source) {
        ...
    }
}
```

Direct manipulation of Ast nodes

```groovy
@GroovyASTTransformation(phase = CompilePhase.INSTRUCTION_SELECTION)
class AddMethodASTTransformation extends AbstractASTTransformation {
    @Override
    void visit(ASTNode[] nodes, SourceUnit source) {
        ClassNode classNode = (ClassNode) nodes[1]

        ReturnStatement code =
                new ReturnStatement(                              ①
                        new ConstantExpression("42"))            ②

        MethodNode methodNode =
                new MethodNode(
                        "getMessage",
                        ACC_PUBLIC,
                        ClassHelper.make(String),
                        [] as Parameter[],
                        [] as ClassNode[],
                        code)                                    ③

        classNode.addMethod(methodNode)                          ④
    }
}
```

More efficient alternatives

GeneralUtils.returnS(callX(c, "m", varX('a')))

new AstBuilder()

.buildFromString()

.buildFromCode()

.buildFromSpec()

Macros

```
.buildFromString ("
          Integer.parseInt("$param")
")
```

```
.buildFromCode (
          Integer.parseInt("$param")
)
```

```
.buildFromSpec {

    method('convertToNumber', ACC_PUBLIC, Integer) {

            parameters { parameter 'parameter': String.class }

            exceptions {}

            block {

                returnStatement {

                    staticMethodCall(Integer, "parseInt") {

                        argumentList {

                            variable "parameter"

                        }

        } } } } }
```

AstSpecificationCompiler

The delegate for the buildFromSpec()
closures

Macro

```
return macro(CompilePhase.SEMANTIC_ANALYSIS, true) {
    return java.security.MessageDigest
            .getInstance('MD5')
            .digest($v { fieldVar }.getBytes())
            .encodeHex()
            .toString()
}
```

Macro

```
@Override
Expression transform(Expression exp) {
    Expression ref = macro { 1 + 1 }

    if (ASTMatcher.matches(ref, exp)) {
        return macro { 3 }
    }

    return super.transform(exp)
}
```

Summary

AST transformations – use and creation

References

http://groovy-lang.org

http://grails.org

http://groovyconsole.appspot.com/

http://www.manning.com/koenig2/