SMDP Exercises 7

Model Driven Development, IT-University of Copenhagen

Niels Søholm - nmar@itu.dk - 2014-03-25

Task 2 - Model transformation (M2M)

```
class NormalizeIdentifiers {

    def static String normalize(String name) {
        // Only allow normal characters, remove any symbols or spaces
        return name.replaceAll("([^a-zA-Z0-9])", "")
    }

    def static void run(FiniteStateMachine it) {
        states.forEach[normalize(name)]
    }
}
```

Task 4 - Code generator specification (M2T)

```
def static compileToJava(Model it) {
    '''
        package JUnitAssert;
        import java.util.Scanner;

        class AssertFramework {
            «FOR assertMethod : assertMethods»
                public boolean «assertMethod.name»(«
                    FOR param : assertMethod.params SEPARATOR ', '»«
                        mapJavaTypes(param.type)» «
                        param.name»«
                    ENDFOR»){
                        return «checkExprType(assertMethod.bodyExpr)»;
                    }
            «ENDFOR»
        }
    '''
}

def static String mapJavaTypes(SimpleTypeEnum type) {
    switch (type) {
        case SimpleTypeEnum.DOUBLE : return type.toString().toLowerCase()
        case SimpleTypeEnum.BOOLEAN : return type.toString().toLowerCase()
        case SimpleTypeEnum.CHAR : return type.toString().toLowerCase()
        case SimpleTypeEnum.INT : return type.toString().toLowerCase()
        case SimpleTypeEnum.SHORT : return type.toString().toLowerCase()
        case SimpleTypeEnum.FLOAT : return type.toString().toLowerCase()
        case SimpleTypeEnum.LONG : return type.toString().toLowerCase()
    }
    return type.toString().toLowerCase().toFirstUpper()
}
```

(continued on next page...)

```
def static String checkExprType(Exp expr) {
    if (expr instanceof BOpMethod) {
        val bopm = (expr as BOpMethod)
        return checkExprType(bopm.lexpr) + "." + bopm.operator + "(" + checkExprType(bopm.rexpr) + ")"
    }
    else if (expr instanceof BOp) {
        val bop = (expr as BOp)
        return "(" + checkExprType(bop.lexpr) + " " + bop.operator + " " + checkExprType(bop.rexpr) + ")"
    }
    else if (expr instanceof UOp) {
        val uop = (expr as UOp)
        return  (uop.operator + checkExprType(uop.expr))
    }
    else if (expr instanceof FunCall) {
        val func = (expr as FunCall)
        var result = (func.name + ".(")
        for (arg : func.arg) {
            result = result + checkExprType(arg) + ", "
        }
        result = result.substring(result.length - 2, 1)
        result = result + ")"
        return result
    }
    else if (expr instanceof Const) {
        return "null"
    }
    else if (expr instanceof Id) {
        val id = (expr as Id)
        return id.name
    }
    else {
        return "error"
    }
}
```