Programmazione Avanzata

Jacopo Notarstefano
jacopo.notarstefano [at] gmail.com

Esercizio 1

Expr.java

```
public abstract class Expr {
        public abstract Expr apply (Expr target, double value);

        public AddExpr add (Expr that) {
                return new AddExpr(this, that);
        }

        public MulExpr mul (Expr that) {
                return new MulExpr(this, that);
        }

        public SubExpr sub (Expr that) {
                return new SubExpr(this, that);
        }
}
```

BinaryExpr.java

```
public abstract class BinaryExpr extends Expr {
        public BinaryExpr (Expr first, Expr second) {
                this.first = first;
                this.second = second;
        }

        public Expr apply (Expr target, double value) {
                this.first = this.first.apply(target, value);
                this.second = this.second.apply(target, value);

                return this;
        }

        protected Expr first;
        protected Expr second;
}
```

AddExpr.java

```
public class AddExpr extends BinaryExpr {
        public AddExpr (Expr first, Expr second) {
                super(first, second);
        }
}
```

MulExpr.java

```
public class MulExpr extends BinaryExpr {
        public MulExpr (Expr first, Expr second) {
                super(first, second);
        }
}
```

SubExpr.java

```
public class SubExpr extends BinaryExpr {
        public SubExpr (Expr first, Expr second) {
                super(first, second);
        }
}
```

UnaryExpr.java

```
public abstract class UnaryExpr extends Expr {
        protected UnaryExpr (Expr argument) {
                this.argument = argument;
        }

        public Expr apply (Expr target, double value) {
                this.argument = this.argument.apply(target, value);

                return this;
        }

        protected Expr argument;
}
```

NegExpr.java

```
public class NegExpr extends UnaryExpr {
        public NegExpr (Expr argument) {
                super(argument);
        }
}
```

ExpExpr.java

```
public class ExpExpr extends UnaryExpr {
        public ExpExpr (Expr argument) {
                super(argument);
```

```
        }
}
```

Function.java

```
public class Function {
        public Function (Expr[] arguments, Expr expression) {
                this.arguments = arguments;
                this.expression = expression;
        }

        public Expr apply (double... values) {
                for (int i = 0; i < values.length; i++) {
                        this.expression = this.expression.apply(arguments[i], values[i]);
                }

                return this.expression;
        }

        private Expr[] arguments;
        private Expr expression;
}
```

Esercizio 2

Esercizio 3

Esercizio 4

Esercizio 5

VectorExpr.java

```
import java.util.List;
import java.util.ArrayList;

public class VectorExpr extends Expr {
        public VectorExpr (int dimension) {
                this.dimension = dimension;
                this.elements = new ArrayList<Expr>(dimension);
        }

        public VectorExpr (int dimension, List<Expr> elements) {
                this.dimension = dimension;
                this.elements = elements;
        }

        public VectorExpr apply (Expr target, double value) {
                for (int i = 0; i < this.dimension; i++) {
                        this.elements.set(i, this.elements.get(i).apply(target, value));
```

```
        }

        return this;
}

public String compile () {
        String result = "";

        result += "double result[] = { ";
        for (int i = 0; i < this.dimension; i++) {
                result += this.elements.get(i).compile();
                result += (i < this.dimension - 1) ? ", " : "";
        }
        result += " }";

        return result;
}

public VectorExpr differentiate (Expr dx) {
        for (int i = 0; i < this.dimension; i++) {
                this.elements.set(i, this.elements.get(i).differentiate(dx));
        }

        return this;
}

public VectorExpr simplify () {
        for (int i = 0; i < this.dimension; i++) {
                this.elements.set(i, this.elements.get(i).simplify());
        }

        return this;
}

public Expr simplify (UnaryExpr argument) {
        return this;
}

public Expr simplify (BinaryExpr parent, Expr sibling) {
        return this;
}

public VectorExpr add (VectorExpr that) {
        List<Expr> newElements = new ArrayList<Expr>(this.dimension);

        for (int i = 0; i < this.dimension; i++) {
                newElements.add(i, new AddExpr(
                        this.elements.get(i),
                        that.elements.get(i)
                ));
        }
```

```
            return new VectorExpr(this.dimension, newElements);
        }

        protected int dimension;
        protected List<Expr> elements;
}
```

Esercizio 6