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 {
        protected 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); }
}
```

DoubleExpr.java

```
public class DoubleExpr extends Expr {
        public DoubleExpr () {}

        public Expr apply (Expr target, double value) {
                return (target == this) ? new ConstExpr(value) : this;
        }
}
```

ConstExpr.java

```
public class ConstExpr extends Expr {
        public ConstExpr (double value) {
                this.value = value;
        }

        public Expr apply (Expr target, double value) {
                return this;
        }

        private double value;
}
```

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

Il codice generato è `((x * 2.0) * exp(y))`.

Expr.java

```
public abstract class Expr {
        // ...

        public abstract String compile ();
}
```

BinaryExpr.java

```
public abstract class BinaryExpr extends Expr {
        // ...

        public String compile () {
                return "(" + this.first.compile() + this.operator() + this.second.compile() + ")";
        }

        protected abstract String operator ();
}
```

AddExpr.java

```
public class AddExpr extends BinaryExpr {
        // ...

        protected String operator () { return " + "; }
}
```

MulExpr.java

```
public class MulExpr extends BinaryExpr {
        // ...

        protected String operator () { return " * "; }
}
```

SubExpr.java

```
public class SubExpr extends BinaryExpr {
        // ...

        protected String operator () { return " - "; }
}
```

UnaryExpr.java

```
public abstract class UnaryExpr extends Expr {
        // ...

        public String compile () {
                return this.operator() + "(" + this.argument.compile() + ")";
        }

        protected abstract String operator ();
}
```

NegExpr.java

```
public class NegExpr extends UnaryExpr {
        // ...

        protected String operator () { return "-"; }
}
```

ExpExpr.java

```
public class ExpExpr extends UnaryExpr {
        // ...

        protected String operator () { return "exp"; }
}
```

DoubleExpr.java

```
public class DoubleExpr extends Expr {
        // ...

        public DoubleExpr (String name) {
                this.name = name;
        }

        public String compile () {
                return this.name;
        }

        private String name;
}
```

ConstExpr.java

```
public class ConstExpr extends Expr {
        // ...

        public String compile () {
                return Double.toString(this.value);
        }
}
```

Esercizio 3

Il codice generato è `((((1.0 * 2.0) + (1.5 * 0.0)) * exp(2.5)) + ((1.5 * 2.0) * (exp(2.5) * 0.0)))`.

Expr.java

```
public abstract class Expr {
        // ...

        public abstract Expr differentiate (Expr dx);
}
```

AddExpr.java

```
public class AddExpr extends BinaryExpr {
        // ...

        public Expr differentiate (Expr dx) {
                return new AddExpr(this.first.differentiate(dx), this.second.differentiate(dx));
        }
}
```

MulExpr.java

```
public class MulExpr extends BinaryExpr {
        // ...

        public Expr differentiate (Expr dx) {
                return new AddExpr(
                        new MulExpr(this.first.differentiate(dx), this.second),
                        new MulExpr(this.first, this.second.differentiate(dx))
                );
        }
}
```

SubExpr.java

```
public class SubExpr extends BinaryExpr {
        // ...

        public Expr differentiate (Expr dx) {
                return new SubExpr(this.first.differentiate(dx), this.second.differentiate(dx));
```

```
        }
}
```

NegExpr.java

```
public class NegExpr extends UnaryExpr {
        // ...

        public Expr differentiate (Expr dx) {
                return new NegExpr(this.argument.differentiate(dx));
        }
}
```

ExpExpr.java

```
public class ExpExpr extends UnaryExpr {
        // ...

        public Expr differentiate (Expr dx) {
                return new MulExpr(new ExpExpr(this.argument), this.argument.differentiate(dx));
        }
}
```

DoubleExpr.java

```
public class DoubleExpr extends Expr {
        // ...

        public Expr differentiate (Expr dx) {
                return (dx == this) ? new ConstExpr(1) : new ConstExpr(0);
        }
}
```

ConstExpr.java

```
public class ConstExpr extends Expr {
        // ...

        public Expr differentiate (Expr dx) {
                return new ConstExpr(0);
        }
}
```

Function.java

```
public class Function {
        // ...

        public Function differentiate (Expr dx) {
                return new Function(this.arguments, this.expression.differentiate(dx));
        }
}
```

Esercizio 4

Il codice ottimizzato generato è `((1.0 * 2.0) * exp(2.5))`.

Expr.java

```
public abstract class Expr {
        // ...

        public abstract Expr simplify ();
        public abstract Expr simplify (UnaryExpr parent);
        public abstract Expr simplify (BinaryExpr parent, Expr sibling);
}
```

BinaryExpr.java

```
public abstract class BinaryExpr extends Expr {
        // ...

        public Expr simplify () {
                this.first = this.first.simplify();
                this.second = this.second.simplify();
                return this.second.simplify(this, this.first);
        }

        public Expr simplify (UnaryExpr parent) { return parent; }
        public Expr simplify (BinaryExpr parent, Expr sibling) { return parent; }
        public abstract Expr simplify (ConstExpr first, Expr second);
}
```

AddExpr.java

```
public class AddExpr extends BinaryExpr {
        // ...

        public Expr simplify (ConstExpr first, Expr second) {
                return first.isZero() ? second : this;
        }
}
```

MulExpr.java

```
public class MulExpr extends BinaryExpr {
        // ...

        public Expr simplify (ConstExpr first, Expr second) {
                return first.isZero() ? first : this;
        }
}
```

SubExpr.java

```
public class SubExpr extends BinaryExpr {
        // ...

        public Expr simplify (ConstExpr first, Expr second) {
                return first.isZero() ? new NegExpr(second) : this;
        }
}
```

UnaryExpr.java

```
public abstract class UnaryExpr extends Expr {
        // ...

        public Expr simplify () {
                this.argument = this.argument.simplify();
                return this.argument.simplify(this);
        }

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

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

NegExpr.java

```
public class NegExpr extends UnaryExpr {
        // ...

        public Expr simplify (NegExpr parent) {
                return parent.simplify(this);
        }
}
```

ExpExpr.java

```
public class ExpExpr extends UnaryExpr {
        // ...

        public Expr simplify (UnaryExpr parent) { return parent; }
}
```

DoubleExpr.java

```
public class DoubleExpr extends Expr {
        // ...

        public Expr simplify () { return this; }
        public Expr simplify (UnaryExpr parent) { return parent; }
```

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

ConstExpr.java

```
public class ConstExpr extends Expr {
        // ...

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

        public boolean isZero () {
                return Math.abs(this.value - 0) < 0.000001;
        }
}
```

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

                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