«enumeration»
ReductionStrategy

NORMAL_ORDER
APPLICATIVE_ORDER
CALL_BY_NAME
CALL_BY_VALUE

LambdaTermVisitor

BetaReductionVisitor

-strategy: ReductionStrategy
-result: LambdaTerm = null
-hasReduced: boolean = false

+BetaReductionVisitor(strategy: ReductionStrategy)
+visit(node: LambdaApplication)
+visit(node: LambdaAbstraction)
+visit(node: LambdaVariable)
+getResult(): LambdaTerm

```
if (hasReduced) {
    result = node;
} else {
    if (node.getFirst().isAbstraction()) {
        LambdaAbstraction abstraction = ((LambdaAbstraction) node.getFirst());
        switch (strategy) {
            case NORMAL_ORDER: {
                result = abstraction.getInside().accept(new ApplicationVisitor(abstraction.getColor(), node.getSecond()));
                break;
            }
            // ...
        }
        hasReduced = true;
    } else {
        node.setFirst(node.getFirst().accept(this));
        node.setSecond(node.getSecond().accept(this));
    }
}
```

```
if (!hasReduced && strategy != CALL_BY_NAME && strategy != CALL_BY_VALUE) {
    node.setInside(node.getInside().accept(this));
}
result = node;
```

```
result = node;
```

«uses»

LambdaTermVisitor

ApplicationVisitor

-color: Color
-replacing: LambdaTerm
-result: LambdaTerm = null

+ApplicationVisitor(color: Color, replacing: LambdaTerm)
+visit(node: LambdaApplication)
+visit(node: LambdaAbstraction)
+visit(node: LambdaVariable)
+getResult(): LambdaTerm

```
node.setFirst(node.getFirst().accept(this));
node.setSecond(node.getSecond().accept(this));
result = node;
```

```
assert(!color.equals(node.getColor()));
node.setInside(node.getInside().accept(this));
result = node;
```

```
result = (color.equals(node.getColor())) ? replacing : node;
```