**package** Command;

**import** Interpreter.Context;

**public** **interface** ExpressionCommand {

**public** Object undo(Context values);

**public** Object evaluate(Context values);

}

**package** Command;

**import** Interpreter.Assignment;

**import** Interpreter.Context;

**public** **class** AssignmentCommand **implements** ExpressionCommand {

**private** Assignment expression;

**public** AssignmentCommand(Assignment expression) {

**this**.expression = expression;

}

@Override

**public** Object undo(Context values) {

values.remove(expression.toString());

**return** **true**;

}

@Override

**public** Object evaluate(Context values) {

**return** expression.evaluate(values);

}

}

**package** Command;

**import** Interpreter.Constant;

**import** Interpreter.Context;

**public** **class** ConstantCommand **implements** ExpressionCommand {

**private** Constant expression;

**public** ConstantCommand(Constant expression) {

**this**.expression = expression;

}

@Override

**public** Object undo(Context values) {

**return** **true**;

}

@Override

**public** Object evaluate(Context values) {

**return** expression.evaluate(values);

}

}

**package** Command;

**import** Interpreter.Context;

**import** Interpreter.DoubleExpression;

**public** **class** DoubleExpressionCommand **implements** ExpressionCommand {

**private** DoubleExpression expression;

**public** DoubleExpressionCommand(DoubleExpression expression) {

**this**.expression = expression;

}

@Override

**public** Object undo(Context values) {

ExpressionCommand firstPart = (ExpressionCommand) expression

.firstPart();

ExpressionCommand nextPart = (ExpressionCommand) expression.nextPart();

firstPart.undo(values);

nextPart.undo(values);

**return** **true**;

}

@Override

**public** Object evaluate(Context values) {

**return** expression.evaluate(values);

}

}

**package** Command;

**import** Interpreter.Constant;

**import** Interpreter.Context;

**import** Interpreter.Move;

**import** Interpreter.Turn;

**public** **class** MoveCommand **implements** ExpressionCommand {

**private** Move expression;

**public** MoveCommand(Move expression) {

**this**.expression = expression;

}

@Override

**public** Object undo(Context values) {

**final** **int** reveredDegrees = 180;

//revere the direction of turtle

Turn reversedDirection = **new** Turn(**new** Constant(-reveredDegrees));

reversedDirection.evaluate(values);

//move in the revered direction by same distance

evaluate(values);

//change the direction to the original direction

Turn originalDirection = **new** Turn(**new** Constant(reveredDegrees));

**return** originalDirection.evaluate(values);

}

@Override

**public** Object evaluate(Context values) {

**return** expression.evaluate(values);

}

}

**package** Command;

**import** Interpreter.Context;

**import** Interpreter.PenDown;

**import** Interpreter.PenUp;

**public** **class** PenDownCommand **implements** ExpressionCommand {

**private** PenDown expression;

**public** PenDownCommand(PenDown expression) {

**this**.expression = expression;

}

@Override

**public** Object undo(Context values) {

PenUp penUpExpression = **new** PenUp();

**return** penUpExpression.evaluate(values);

}

@Override

**public** Object evaluate(Context values) {

**return** expression.evaluate(values);

}

}

**package** Command;

**import** Interpreter.Context;

**import** Interpreter.PenDown;

**import** Interpreter.PenUp;

**public** **class** PenUpCommand **implements** ExpressionCommand {

**private** PenUp expression;

**public** PenUpCommand(PenUp expression) {

**this**.expression = expression;

}

@Override

**public** Object undo(Context values) {

PenDown penDownExpression = **new** PenDown();

**return** penDownExpression.evaluate(values);

}

@Override

**public** Object evaluate(Context values) {

**return** expression.evaluate(values);

}

}

**package** Command;

**import** Interpreter.Context;

**import** Interpreter.Repeat;

**public** **class** RepeatCommand **implements** ExpressionCommand {

**private** Repeat expression;

**public** RepeatCommand(Repeat expression) {

**this**.expression = expression;

}

@Override

**public** Object undo(Context values) {

**int** parameter = expression.counter(values);

ExpressionCommand statement =

(ExpressionCommand) expression.statement();

**for** (**int** i = 0; i < parameter; i++)

statement.undo(values);

**return** **true**;

}

@Override

**public** Object evaluate(Context values) {

**return** expression.evaluate(values);

}

}

**package** Command;

**import** Interpreter.Constant;

**import** Interpreter.Context;

**import** Interpreter.Expression;

**import** Interpreter.Turn;

**public** **class** TurnCommand **implements** ExpressionCommand {

**private** Turn expression;

**public** TurnCommand(Turn expression) {

**this**.expression = expression;

}

@Override

**public** Object undo(Context values) {

Expression parameter = expression.parameter();

**int** degrees = (Integer) parameter.evaluate(values);

Constant reveresDegrees = **new** Constant(-degrees);

Turn reverse = **new** Turn(reveresDegrees);

**return** reverse.evaluate(values);

}

@Override

**public** Object evaluate(Context values) {

**return** expression.evaluate(values);

}

}

**package** Command;

**import** Interpreter.Context;

**import** Interpreter.Variable;

**public** **class** VariableCommand **implements** ExpressionCommand {

**private** Variable expression;

**public** VariableCommand(Variable expression) {

**this**.expression = expression;

}

@Override

**public** Object undo(Context values) {

String name = expression.toString();

values.remove(name);

**return** **true**;

}

@Override

**public** Object evaluate(Context values) {

**return** expression.evaluate(values);

}

}

**package** Command;

**import** **static** org.junit.Assert.\*;

**import** org.junit.After;

**import** org.junit.Before;

**import** org.junit.Test;

**import** Interpreter.Constant;

**import** Interpreter.Context;

**import** Interpreter.Move;

**import** Interpreter.PenDown;

**import** Interpreter.PenUp;

**import** Interpreter.Turn;

**import** Turtle.Turtle;

**public** **class** UndoTest {

**private** Context values;

**private** **static** **final** **double** *DELTA* = 1e-3;// degree of precision used in comparing doubles

@Before

**public** **void** setUp() **throws** Exception {

values = **new** Context();

}

@After

**public** **void** tearDown() **throws** Exception {

}

@Test

**public** **void** testUndoPenUp() {

PenUp expression = **new** PenUp();

ExpressionCommand penUpCommand = **new** PenUpCommand(expression);

penUpCommand.evaluate(values);

Turtle testTurtle = values.turtle();

*assertTrue*(testTurtle.isPenUp());

penUpCommand.undo(values);

*assertFalse*(testTurtle.isPenUp());

}

@Test

**public** **void** testUndoPenDown() {

PenDown expression = **new** PenDown();

ExpressionCommand penDownCommand = **new** PenDownCommand(expression);

penDownCommand.evaluate(values);

Turtle testTurtle = values.turtle();

*assertFalse*(testTurtle.isPenUp());

penDownCommand.undo(values);

*assertTrue*(testTurtle.isPenUp());

}

@Test

**public** **void** testUndoMove() {

Move expression = **new** Move(**new** Constant(10));

ExpressionCommand moveCommand = **new** MoveCommand(expression);

moveCommand.evaluate(values);

Turtle testTurtle = values.turtle();

*assertEquals*(0, testTurtle.direction());

*assertEquals*(10, testTurtle.location().getX(), *DELTA*);

*assertEquals*(0, testTurtle.location().getY(), *DELTA*);

moveCommand.undo(values);

//compare with original state of turtle

//original state is at direction 0 degrees and location at x=0 and y=0

*assertEquals*(0, testTurtle.direction());

*assertEquals*(0, testTurtle.location().getX(), *DELTA*);

*assertEquals*(0, testTurtle.location().getY(), *DELTA*);

}

@Test

**public** **void** testUndoTurn() {

Turn expression = **new** Turn(**new** Constant(90));

ExpressionCommand turnCommand = **new** TurnCommand(expression);

turnCommand.evaluate(values);

Turtle testTurtle = values.turtle();

*assertEquals*(90, testTurtle.direction());

*assertEquals*(0, testTurtle.location().getX(), *DELTA*);

*assertEquals*(0, testTurtle.location().getY(), *DELTA*);

turnCommand.undo(values);

//compare with original state of turtle

//original state is at direction 0 degrees and location at x=0 and y=0

*assertEquals*(0, testTurtle.direction());

*assertEquals*(0, testTurtle.location().getX(), *DELTA*);

*assertEquals*(0, testTurtle.location().getY(), *DELTA*);

}

}