**Testing Number and Variable Class**

**Testing Number class:**

1. Create Number object with value
2. Use toString and it should return the value.
3. Create two other Number Objects, one with the same value, one with a different value
4. Use equals on both new Number objects. The one with the same value should return true. The other should return false.
5. Use the value method that takes an input on the first Number object. Should return the value.
6. Use the value method that takes no input on the first Number object. Should return the value.
7. Use derivative on the first Number object. Should return 0.

> Number a = new Number(5.0)

> a.toString()

"5.0"

> Number a2 = new Number(5.0)

> Number b = new Number(7.0)

> a.equals(a2)

true

> a.equals(b)

false

> a.value(3.0)

5.0

> a.value()

5.0

**Testing Variable class:**

1. Create Variable object with no value
2. Use toString and it should return x.
3. Use equals on the Variable object. This should return true.
4. Use the value method that takes an input on the first Variable object. Should return the value.
5. Use the value method that takes no input on the first Variabel object. This should throw an exception.
6. Use derivative on the first Variable object. Should return 1.

> Variable c = new Variable()

> c.toString()

"x"

> c.equals()

true

> c.value(5.0)

5.0

> c.value()

java.lang.UnsupportedOperationException

at Variable.value(Variable.java:22)

> c.derivative()

1.0

**Tested using the Number class for all function objects**

Equals method is not returning the right value for when objects are equal.

**Testing Cos class:**

1. Create Cos object with another Number object for a value.
2. Use toString and it should return the Cos(value of the function).
3. Create two other Cos Objects, one with the same value, one with a different value.
4. Use equals on both new cos objects. The one with the same value should return true. The other should return false.
5. Use the value method that takes an input on the first Cos object. Should return the value evaluated for cos.
6. Use the value method that takes no input on the first Cos object. Should return the value returned from the value() function of this.operand.
7. Use derivative on the first cos object. Should return –sin(operand).

> Cos d = new Cos(new Number(5.0))

> Cos d2 = new Cos(new Number(5.0))

> Cos e = new Cos(new Number(7.0))

> d.getOperand()

5.0

> d.toString()

"cos(5.0)"

> d.equals(d2)

false

> d.equals(e)

false

> d.value(5.0)

0.28366218546322625

> d.value()

0.28366218546322625

> d.derivative()

-1.0 \* Sin(5.0)

**Testing Sin class:**

1. Create Sin object with a Number object for a value.
2. Use toString and it should return the Sin(value of the function).
3. Create two other Sin Objects, one with the same value, one with a different value.
4. Use equals on both new Sin objects. The one with the same value should return true. The other should return false.
5. Use the value method that takes an input on the first Sin object. Should return the value evaluated for Sin.
6. Use the value method that takes no input on the first Sin object. Should return the value returned from the value() function of this.operand.
7. Use derivative on the first sin object. Should return cos (operand).

> Sin f = new Sin(new Number(5.0))

> Sin f2 = new Sin(new Number(5.0))

> Sin g = new Sin(new Number(7.0))

> f.getOperand()

5.0

> f.equals(f2)

false

> f.equals(g)

false

> f.toString()

"Sin(5.0)"

> f.value(5.0)

-0.9589242746631385

> f.value()

-0.9589242746631385

> f.derivative()

cos(5.0)

**Testing Exp class:**

1. Create Exp object with a Number object for a value.
2. Use toString and it should return the Exp(value of the function).
3. Create two other Exp Objects, one with the same value, one with a different value.
4. Use equals on both new Exp objects. The one with the same value should return true. The other should return false.
5. Use the value method that takes an input on the first Exp object. Should return the value evaluated for Exp.
6. Use the value method that takes no input on the first Exp object. Should return the value returned from the value() function of this.operand.
7. Use derivative on the first Exp object. Should return operand.derivative\*cos (operand).

> Exp h = new Exp(new Number(5.0))

> Exp h2 = new Exp(new Number(5.0))

> Exp i = new Exp(new Number(5.0))

> h.getOperand()

5.0

> h.equals(h2)

false

> h.equals(i)

false

> h.toString()

"Exp(5.0)"

> h.value(5.0)

148.4131591025766

> h.value()

148.4131591025766

> h.derivative()

0.0 \* Exp(5.0)

**Testing Polynomial Class**

1. Create Polynomial object with a Number object for a value and a double for an exponent.
2. Use toString and it should return the function ^ exponent value.
3. Create two other Polynomial Objects, one with the same value, one with a different value.
4. Use equals on both new Polynomial objects. The one with the same value should return true. The other should return false.
5. Use the value method that takes an input on the first Polynomial object. Should return the value evaluated for Polynomial.
6. Use the value method that takes no input on the first Polynomial object. Should return the value returned from the value() function of this.operand.
7. Use derivative on the first Polynomial object. Should power\*function^(power-1)

> Polynomial j = new Polynomial(new Number(5.0),2.0)

> Polynomial j2 = new Polynomial(new Number(5.0),2.0)

> Polynomial k = new Polynomial(new Number(5.0),2.0)

> j.getPower()

2.0

> j.getOperand()

5.0

> j.equals(j2)

false

> j.equals(k)

false

> j.value(7.0)

25.0

> j.value()

25.0

> j.derivative()

2.0 \* (5.0^1.0)

> j.toString()

"(5.0^2.0)"

**Testing binaryOp class:**

1. Make six binaryOps: 5 for the operators, put 2 function into operator and a copy of the first.
2. Use getOperator on first binaryOp should return the sign it was initialized with.
3. Use getOperand on first binaryOp: should return null.
4. Use getLeftOperand: should return the left operand that was initialized with.
5. Use getRightOperand: should return the right operand that was initialized with.
6. Use the equals method with the two identical operators and uses it with two different binaryOp.
7. Use derivative on all five binary operators should return the appropriate derivative.
8. Use the toString on the five main binaryOps should return (left char right).
9. Use the value with input on the five main binaryOps. Should return left and right evaluated in terms of the input. Then the values added together.
10. Use value with no input. Should return left and right evaluated with no input the add the two values together

> Number num = new Number(5.0)

> Number num2 = new Number(7.0)

> binaryOp l = new binaryOp('+', num, num2)

> binaryOp m = new binaryOp('-', num, num2)

> binaryOp n = new binaryOp('\*', num, num2)

> binaryOp o = new binaryOp('/', num, num2)

> binaryOp p = new binaryOp('^', num, num2)

> binaryOp l2 = new binaryOp('^', num, num2)

> l.getOperator()

'+'

> l.getLeftOperand()

5.0

> l.getRightOperand()

7.0

> l.equals(l2)

false

> l.equals(m)

false

> l.derivative()

0.0 + 0.0

> m.derivative()

0.0 - 0.0

> n.derivative()

(5.0 \* 0.0) + 0.0 \* 7.0

> o.derivative()

((7.0 \* 0.0) - 0.0 \* 5.0) / 7.0 \* 2.0

> p.derivative()

7.0 \* (5.0 ^ (7.0 - 1.0))

> l.toString()

"5.0 + 7.0"

> m.toString()

"5.0 - 7.0"

> n.toString()

"5.0 \* 7.0"

> o.toString()

"5.0 / 7.0"

> p.toString()

"5.0 ^ 7.0"

> l.value(3.0)

12.0

> m.value(3.0)

-2.0

> n.value(3.0)

35.0

> o.value(3.0)

0.7142857142857143

> p.value(3.0)

78125.0

> l.value()

12.0

> m.value()

-2.0

> n.value()

35.0

> o.value()

0.7142857142857143

> p.value()

78125.0

**Tested using the Number class for all function objects**

Equals method is not returning the right value for when objects are equal.

**Testing Cos class:**

1. Create Cos object with another variable for a function.
2. Use toString and it should return the Cos(x).
3. Create one other Cos Objects.
4. Use equals on both new cos objects. This should return true.
5. Use the value method that takes an input on the first Cos object. Should return the value evaluated for cos.
6. Use the value method that takes no input on the first Cos object. Should throw an exception.
7. Use derivative on the first cos object. Should return –sin(x).

> Cos q = new Cos(new Variable())

> Cos q2 = new Cos(new Variable())

> q.toString()

"cos(x)"

> q.equals(q2)

false

> q.value(5.0)

0.28366218546322625

> q.value()

java.lang.UnsupportedOperationException

at Variable.value(Variable.java:27)

at Cos.value(Cos.java:34)

> q.derivative()

-1.0 \* Sin(x)

**Testing Sin class:**

1. Create Sin object with another variable for a function.
2. Use toString and it should return the Sin(x).
3. Create one other Sin Objects.
4. Use equals on both new sin objects. This should return true.
5. Use the value method that takes an input on the first Sin object. Should return the value evaluated for sin.
6. Use the value method that takes no input on the first Sin object. Should throw an exception.
7. Use derivative on the first sin object. Should return cos (x).

> Sin r = new Sin(new Variable())

> Sin r2 = new Sin(new Variable())

> r.toString()

"Sin(x)"

> r.equals(r2)

false

> r.value(5.0)

-0.9589242746631385

> r.value()

java.lang.UnsupportedOperationException

at Variable.value(Variable.java:27)

at Sin.value(Sin.java:33)

> r.derivative()

cos(x)

**Testing Exp class:**

1. Create Exp object with another variable for a function.
2. Use toString and it should return the Exp(x).
3. Create one other Exp Objects.
4. Use equals on both new Exp objects. This should return true.
5. Use the value method that takes an input on the first Exp object. Should return the value evaluated for Exp.
6. Use the value method that takes no input on the first Exp object. Should throw an exception.
7. Use derivative on the first Exp object. Should return operand.derivative\*Exp^ (operand ).

> Exp s = new Exp(new Variable())

> Exp s2 = new Exp(new Variable())

> s.toString()

"Exp(x)"

> s.equals(s2)

false

> s.value(5.0)

148.4131591025766

> s.value()

java.lang.UnsupportedOperationException

at Variable.value(Variable.java:27)

at Exp.value(Exp.java:33)

> s.derivative()

1.0 \* Exp(x)

**Testing Polynomial Class**

1. Create Polynomial object with a Variable object for a value and a double for an exponent.
2. Use toString and it should return the x ^ exponent value.
3. Create one other Polynomial Objects.
4. Use equals on new Polynomial objects. This should return true.
5. Use the value method that takes an input on the first Polynomial object. Should return the value evaluated for Polynomial.
6. Use the value method that takes no input on the first Polynomial object. Should throw an exception.
7. Use derivative on the first Polynomial object. Should power\*function^(power-1)\*function.derivative.

> Polynomial t = new Polynomial(new Variable(), 5.0)

> Polynomial t2 = new Polynomial(new Variable(), 5.0)

> t.toString()

"(x^5.0)"

> t.equals(t2)

false

> t.value(2.0)

32.0

> t.value()

java.lang.UnsupportedOperationException

at Variable.value(Variable.java:27)

at Polynomial.value(Polynomial.java:36)

> t.derivative()

(5.0 \* (x^4.0)) \* 1.0

**Testing binaryOp class:**

1. Make six binaryOps: 5 for the operators, put 2 Variable objects into operator and a copy of the first.
2. Use getOperator on first binaryOp should return the sign it was initialized with.
3. Use getOperand on first binaryOp: should return null.
4. Use getLeftOperand: should return the left operand that was initialized with.
5. Use getRightOperand: should return the right operand that was initialized with.
6. Use the equals method with the two identical operators and uses it with two different binaryOp.
7. Use derivative on all five binary operators should return the appropriate derivative.
8. Use the toString on the five main binaryOps should return (left char right).
9. Use the value with input on the five main binaryOps. Should return left and right evaluated in terms of the input. Then the values added together.
10. Use value with no input. Should return left and right evaluated with no input the add the two values together

> Variable var = new Variable()

> Variable var2 = new Variable()

> binaryOp u = new binaryOp('+', var, var2)

> binaryOp v = new binaryOp('-', var, var2)

> binaryOp w = new binaryOp('\*', var, var2)

> binaryOp x = new binaryOp('/', var, var2)

> binaryOp y = new binaryOp('^', var, var2)

> binaryOp z = new binaryOp('+', var, var2)

> u.getOperator()

'+'

> u.getOperand()

null

> u.getLeftOperand()

x

> u.getRightOperand()

x

> u.equals(z)

true

> u.equals(v)

false

> u.derivative()

1.0 + 1.0

> v.derivative()

1.0 - 1.0

> w.derivative()

(x \* 1.0) + 1.0 \* x

> x.derivative()

((x \* 1.0) - 1.0 \* x) / x \* 2.0

> y.derivative()

x \* (x ^ (x - 1.0))

> u.toString()

"x + x"

> v.toString()

"x - x"

> w.toString()

"x \* x"

> x.toString()

"x / x"

> y.toString()

"x ^ x"

> u.value(2.0)

4.0

> v.value(2.0)

0.0

> w.value(2.0)

4.0

> x.value(2.0)

1.0

> y.value(2.0)

4.0

> u.value()

java.lang.UnsupportedOperationException

at Variable.value(Variable.java:27)

at binaryOp.value(binaryOp.java:100)

> v.value()

java.lang.UnsupportedOperationException

at Variable.value(Variable.java:27)

at binaryOp.value(binaryOp.java:102)

> w.value()

java.lang.UnsupportedOperationException

at Variable.value(Variable.java:27)

at binaryOp.value(binaryOp.java:104)

> x.value()

java.lang.UnsupportedOperationException

at Variable.value(Variable.java:27)

at binaryOp.value(binaryOp.java:106)

> y.value()

java.lang.UnsupportedOperationException

at Variable.value(Variable.java:27)

at binaryOp.value(binaryOp.java:108)