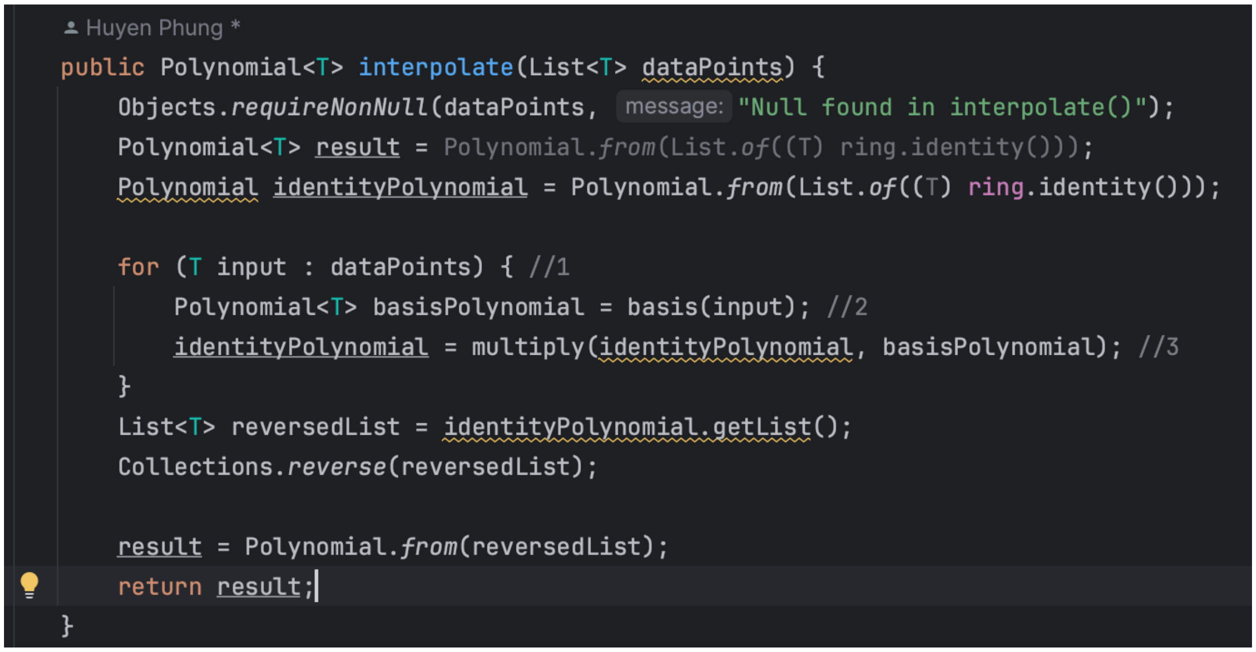
-
Name: Harley Phung
Class: CSDS 293

Test design document

Class under test: *Interpolate*

Method under test: *interpolate(List dataPoints)*



```
Huyen Phung *
public Polynomial<T> interpolate(List<T> dataPoints) {
    Objects.requireNonNull(dataPoints, message: "Null found in interpolate()");
    Polynomial<T> result = Polynomial.from(List.of((T) ring.identity()));
    Polynomial identityPolynomial = Polynomial.from(List.of((T) ring.identity()));

    for (T input : dataPoints) { //1
        Polynomial<T> basisPolynomial = basis(input); //2
        identityPolynomial = multiply(identityPolynomial, basisPolynomial); //3
    }
    List<T> reversedList = identityPolynomial.getList();
    Collections.reverse(reversedList);

    result = Polynomial.from(reversedList);
    return result;
}
```

Test Conditions:

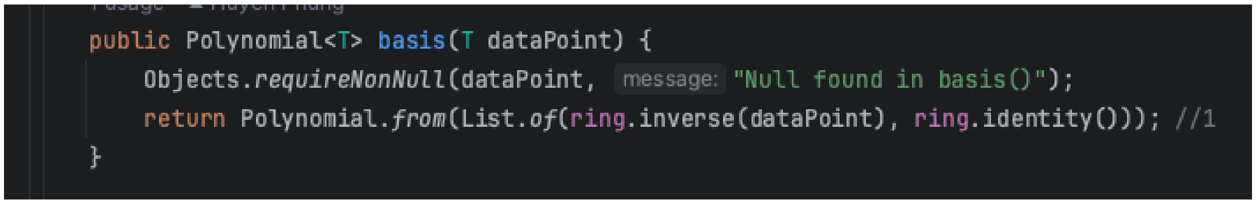
Identifier	Goal	Notes	Condition
CC1	Code Coverage	1	dataPoints is not empty
BC1	Branch Coverage	1	dataPoints has remaining in list
B1	Boundary		Current input index = dataPoints.get(index)
B2	Boundary		Current input index != dataPoints.get(index)
BD1	Bad Data		dataPoints is empty

Tests:

Test	Test condition	Conditions satisfied	Assertion

1	dataPoints is null	BD1	
2	dataPoints in not null	CC1	Current index in range
3	dataPoints not empty	CC1, BC1	dataPoints != ∅
4	dataPoints empty	CC1, BC2	dataPoints = ∅

Method under test: *basis(dataPoint)*



```
public Polynomial<T> basis(T dataPoint) {
    Objects.requireNonNull(dataPoint,  message: "Null found in basis()");
    return Polynomial.from(List.of(ring.inverse(dataPoint), ring.identity())); //1
}
```

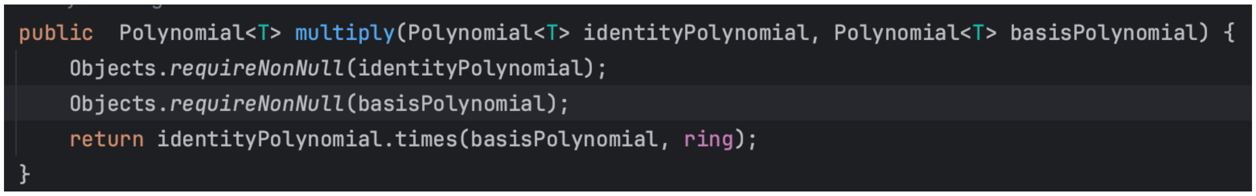
Condition

Identifier	Goal	Notes	Condition
CC1	Code Coverage		dataPoints not null
BD1	Bad Data		dataPoints null

Tests:

Test	Test condition	Conditions satisfied	Assertion
1	dataPoints is not null	CC1	dataPoints != ∅
2	dataPoints is null	BD1	dataPoints = ∅

Method under test: *multiply (Polynomial identityPolynomial, Polynomial basisPolynomial)*



```java
public  Polynomial<T> multiply(Polynomial<T> identityPolynomial, Polynomial<T> basisPolynomial) {
    Objects.requireNonNull(identityPolynomial);
    Objects.requireNonNull(basisPolynomial);
    return identityPolynomial.times(basisPolynomial, ring);
}
```

Conditions:

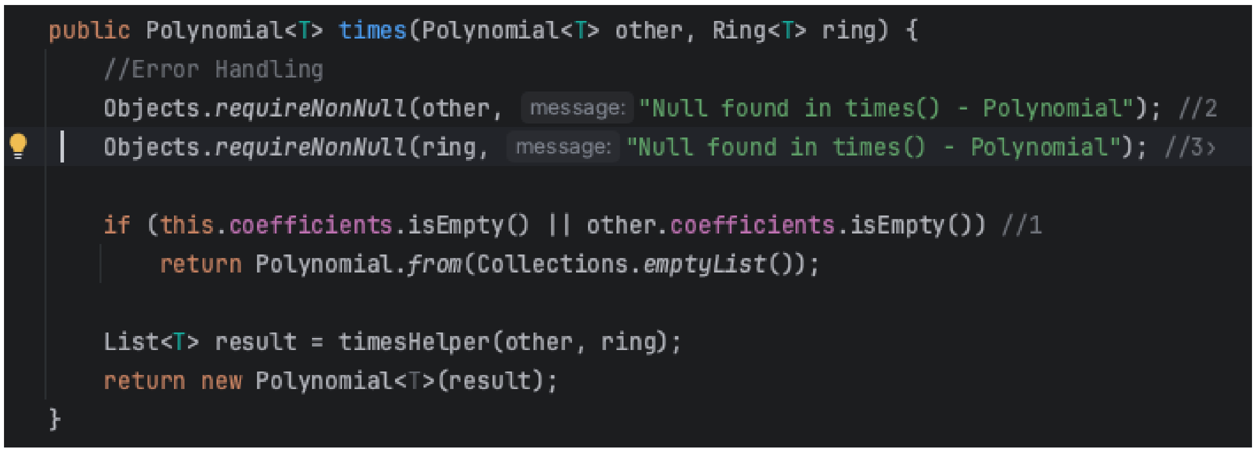
Identifier	Goal	Notes	Condition
CC1	Code Coverage	1	identityPolyonomial is not null/empty
CC2	Code Coverage	2	basisPolynomial is not null/empty
BD1	Bad Data	1	identityPolynomial is null
BD2	Bad Data	2	basisPolynomial is null
BD3	Bad Data	3	identityPolynomial is empty
BD4	Bad Data	3	basisPolynomial is empty

Test:

Test	Test condition	Conditions satisfied	Assertion
1	identityPolyonimal is null	BD1	Exception
2	basisPolyonimal is null	BD2	Exception
3	identityPolynomial and basisPolyonimal are not null	CC1, CC2, BD1, BD2	basis != ∅

Class under test: *Polynomial*

Method under test: *times(Polynomial otherPolynomial, Ring ring)*



```
public Polynomial<T> times(Polynomial<T> other, Ring<T> ring) {
    //Error Handling
    Objects.requireNonNull(other, message: "Null found in times() - Polynomial"); //2
    Objects.requireNonNull(ring, message: "Null found in times() - Polynomial"); //3

    if (this.coefficients.isEmpty() || other.coefficients.isEmpty()) //1
        return Polynomial.from(Collections.emptyList());

    List<T> result = timesHelper(other, ring);
    return new Polynomial<T>(result);
}
```

Conditions:

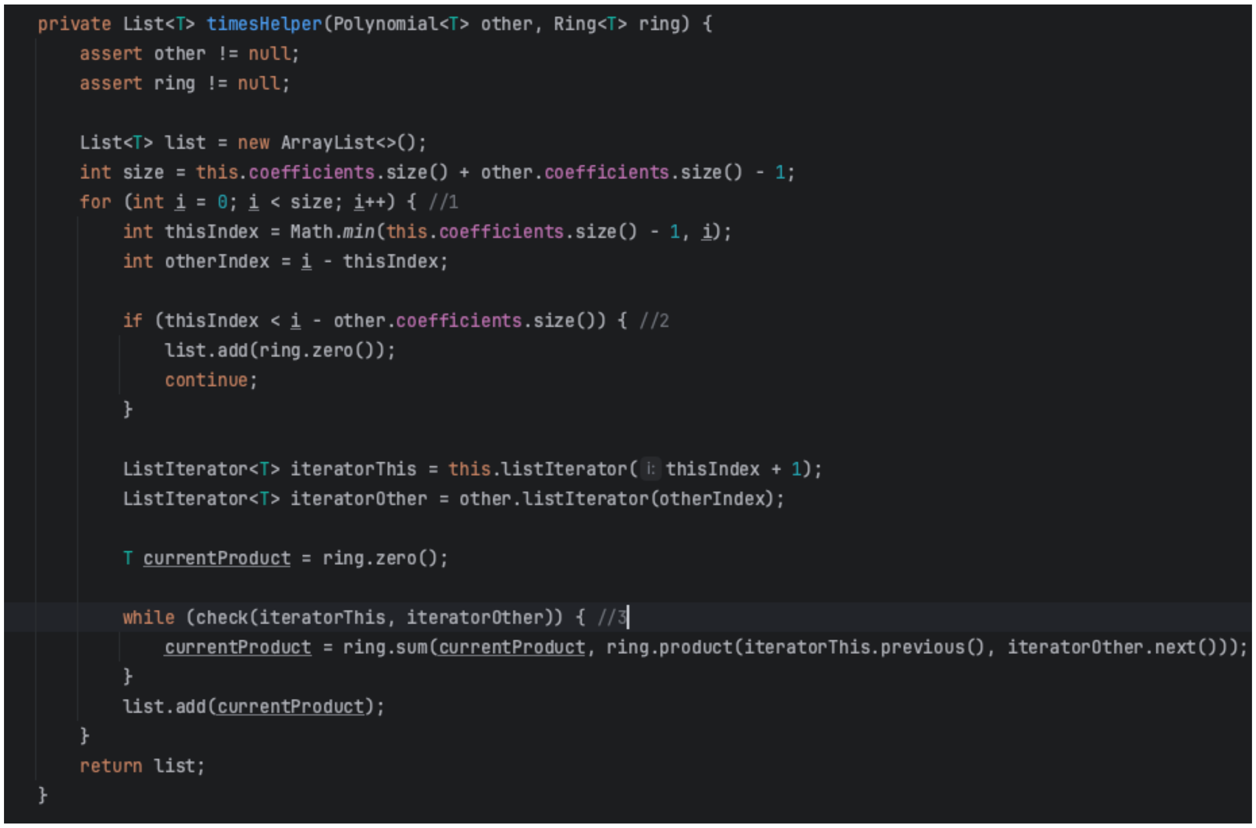
Identifier	Goal	Notes	Condition
CC1	Code Coverage	1	Check if there's any coefficient list is empty
BR1	Branch Coverage	1.1	If current checked polynomial coefficient is empty
BR2	Branch Coverage	1.2	If otherPolynomial coefficient is empty
BR3	Branch Coverage	1.3	If current checked polynomial coefficient is not empty and otherPolynomial coefficient is not empty
BD1	Bad Data	2	If otherPolynomial is null
BD2	Bad Data	3	If ring is null

Test:

Test	Test condition	Conditions satisfied	Assertion
1	otherPolynomial is null	BD1	Exception

2	Ring is null	BD2	Exception
3	This polynomial coefficients is empty	BR1	New empty coefficient list polynomial
4	otherPolynomial coefficients is empty	BR2	New empty coefficient list polynomial
5	This coefficient list and otherPolynomial list is not empty	BR3	New polynomial that contains the list of coefficient that follows the times rule of polynomial

Method under test: *timesHelper(Polynomial other, Ring ring)*



```java
private List<T> timesHelper(Polynomial<T> other, Ring<T> ring) {
    assert other != null;
    assert ring != null;

    List<T> list = new ArrayList<>();
    int size = this.coefficients.size() + other.coefficients.size() - 1;
    for (int i = 0; i < size; i++) { //1
        int thisIndex = Math.min(this.coefficients.size() - 1, i);
        int otherIndex = i - thisIndex;

        if (thisIndex < i - other.coefficients.size()) { //2
            list.add(ring.zero());
            continue;
        }

        ListIterator<T> iteratorThis = this.listIterator( i: thisIndex + 1);
        ListIterator<T> iteratorOther = other.listIterator(otherIndex);

        T currentProduct = ring.zero();

        while (check(iteratorThis, iteratorOther)) { //3
            currentProduct = ring.sum(currentProduct, ring.product(iteratorThis.previous(), iteratorOther.next()));
        }
        list.add(currentProduct);
    }
    return list;
}
```

Conditions:

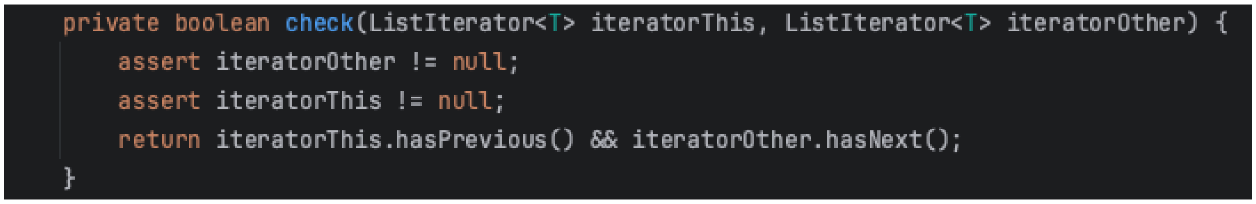
Identifier	Goal	Notes	Condition
CC1	Code Coverage	1	Size of returned polynomial's coefficient list is not 0
BR1	Branch Coverage	1 true	Variable i is in range 0 to size
BR2	Branch Coverage	1 false	Variable i is out of range
BR3	Branch Coverage	2 true	If the current checked polynomial's coefficient list has smaller size the other.
BR4	Branch Coverage	2 false	If the current checked polynomial's coefficient list has similar of larger size the other.
BR5	Branch Coverage	3 true	If the condition to do product is passed
BR6	Branch Coverage	3 false	If the condition to do product is false
B1	Boundary	1, 2 true	This polynomial size is smaller than other.
B2	Boundary	1, 2 false	This polynomial size is larger or equal than other
B3	Boundary	1,3 true	Can move to next index
B4	Boundary	1,3 false	Cannot move to next index
BD1	Bad data		Other is null
BD2	Bad data		Ring is null

Tests:

Test	Test condition	Conditions satisfied	Assertion
1	0 <= i < size	BR1	
2	i >= size	BR2	Empty ArrayList
3	This.size < other.size	BR3, B3, B1	

4	this.size >= other.size	BR4, B4, B2	
5	this polynomial is iterable. other polynomial is iterable	BR5, B3	Calculate the product between 2 polynomials. Move to next index of these polynomial
6	this polynomial is not iterable.	BR6, B4	Return list of coefficients
7	other polynomial is not iterable	BR6, B4	Return list of coefficients

Method under test: check(Polynomial other, Ring ring)



```
private boolean check(ListIterator<T> iteratorThis, ListIterator<T> iteratorOther) {
    assert iteratorOther != null;
    assert iteratorThis != null;
    return iteratorThis.hasPrevious() && iteratorOther.hasNext();
}
```

Conditions:

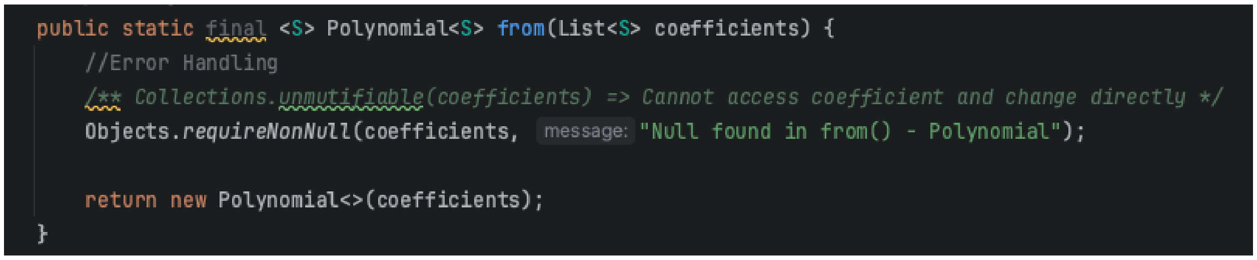
Identifier	Goal	Notes	Condition
CC1	Code Coverage		Return Exception or boolean value
BD1	Bad Data		iteratorThis is null => assert Exception
BD2	Bad Data		iteratorOther is null => assert Exception

Tests:

Test	Test condition	Conditions satisfied	Assertion
1	iteratorThis null	BD1, CC1	Exception
2	iteratorOther null	BD2, CC1	Exception

3	iteratorThis have previous index and iteratorOther has next index	CC1	Boolean value

Method under test: from(List coefficients)



```java
public static final <S> Polynomial<S> from(List<S> coefficients) {
    //Error Handling
    /** Collections.unmutifiable(coefficients) => Cannot access coefficient and change directly */
    Objects.requireNonNull(coefficients,  message: "Null found in from() - Polynomial");

    return new Polynomial<>(coefficients);
}
```

Conditions:

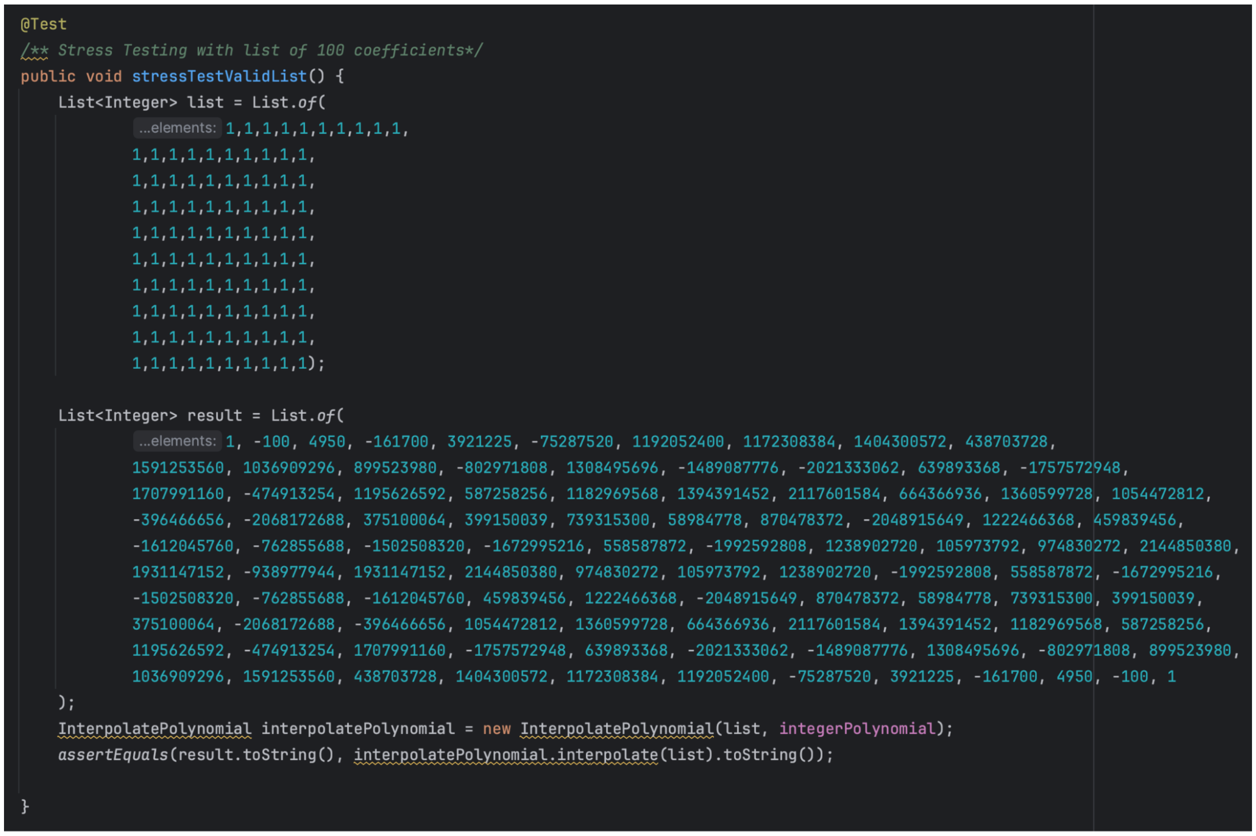
Identifier	Goal	Notes	Condition
CC1	Code Coverage		Return Exception or new Polynomial
BD1	Bad Data		Coefficients is null

Tests:

Test	Test condition	Conditions satisfied	Assertion
1	Coefficients is null	BD1, CC1	Exception
2	Coefficients is not null	CC1	New Polynomial with assigned coefficients

Stress Test:

Method under test: interpolate(List dataPoints)



```java
@Test
/** Stress Testing with list of 100 coefficients*/
public void stressTestValidList() {
    List<Integer> list = List.of(
            ...elements: 1,1,1,1,1,1,1,1,1,1,
            1,1,1,1,1,1,1,1,1,1,
            1,1,1,1,1,1,1,1,1,1,
            1,1,1,1,1,1,1,1,1,1,
            1,1,1,1,1,1,1,1,1,1,
            1,1,1,1,1,1,1,1,1,1,
            1,1,1,1,1,1,1,1,1,1,
            1,1,1,1,1,1,1,1,1,1,
            1,1,1,1,1,1,1,1,1,1,
            1,1,1,1,1,1,1,1,1,1);

    List<Integer> result = List.of(
            ...elements: 1, -100, 4950, -161700, 3921225, -75287520, 1192052400, 1172308384, 1404300572, 438703728,
            1591253560, 1036909296, 899523980, -802971808, 1308495696, -1489087776, -2021333062, 639893368, -1757572948,
            1707991160, -474913254, 1195626592, 587258256, 1182969568, 1394391452, 2117601584, 664366936, 1360599728, 1054472812,
            -396466656, -2068172688, 375100064, 399150039, 739315300, 58984778, 870478372, -2048915649, 1222466368, 459839456,
            -1612045760, -762855688, -1502508320, -1672995216, 558587872, -1992592808, 1238902720, 105973792, 974830272, 2144850380,
            1931147152, -938977944, 1931147152, 2144850380, 974830272, 105973792, 1238902720, -1992592808, 558587872, -1672995216,
            -1502508320, -762855688, -1612045760, 459839456, 1222466368, -2048915649, 870478372, 58984778, 739315300, 399150039,
            375100064, -2068172688, -396466656, 1054472812, 1360599728, 664366936, 2117601584, 1394391452, 1182969568, 587258256,
            1195626592, -474913254, 1707991160, -1757572948, 639893368, -2021333062, -1489087776, 1308495696, -802971808, 899523980,
            1036909296, 1591253560, 438703728, 1404300572, 1172308384, 1192052400, -75287520, 3921225, -161700, 4950, -100, 1
    );
    InterpolatePolynomial interpolatePolynomial = new InterpolatePolynomial(list, integerPolynomial);
    assertEquals(result.toString(), interpolatePolynomial.interpolate(list).toString());

}
```

- Having the list of dataPoints of 100 coefficients.
- Applied interpolate(list of dataPoints)
- Return result.