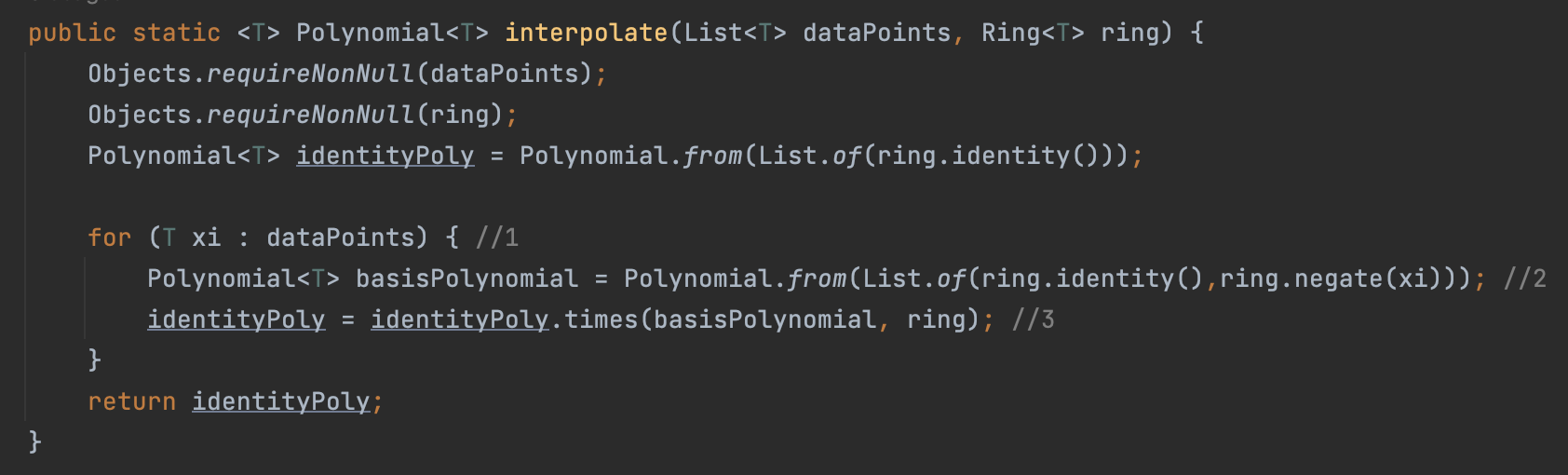
Name: Helen Nguyen

Case ID: hxn150

**Test Design Document**

**Tested Class:** Interpolation

**Method: interpolate(List<T> dataPoints, Ring<T> ring)**

****

Test Conditions:

| Identifier | Goal | Notes | Condition |
| --- | --- | --- | --- |
| CC1 | Code Coverage | 1 | dataPoints is not empty |
| BC1 | Branch Coverage | 1 | dataPoints remain in list |
| BC2 | Branch Coverage | 1 | dataPoints has no remaining in the list |
| BD1 | Bad Data |  | dataPoints is empty |
| B1 | Boundary |  | Current input index = index of dataPoints |
| B1 | Boundary |  | Current input index != index of dataPoints |

Tests:

| Test No. | Test | Test Condition | Conditions Satisfied | Assertion |
| --- | --- | --- | --- | --- |
| 1 | testCodeCoverage | dataPoints is not null (CC1) | dataPoints not empty | Polynomial result is not null |
| 2 | testBranchCoverageEmptyDataPoints | dataPoints is not empty (CC1, BC1) | dataPoints = ∅ | Polynomial result is not null |
| 3 | testDataFlow | dataPoints is not null (CC1) | dataPoints not empty | Polynomial result is not null |
| 4 | testBoundaryCases | dataPoints is not null (CC1) | dataPoints not empty | Polynomial result is not null |
| 5 | testCompoundBoundaries | dataPoints is not null (CC1) | dataPoints not empty | Polynomial result is not null |
| 6 | testBadDataNullDataPoints | dataPoints is null (BD1) | dataPoints = ∅ | Expected NullPointerException |
| 7 | testBadDataNullRing | dataPoints is not null (CC1) | dataPoints not empty | Expected NullPointerException |
| 8 | testGoodData | dataPoints is not null (CC1) | dataPoints not empty | Polynomial result is not null |
| 9 | testPlusDifferentSize | dataPoints is not null (CC1) | dataPoints not empty | Polynomial addition result is as expected |
| 10 | testPlusEmptyAndNonEmpty | dataPoints is not null (CC1) | dataPoints not empty | Polynomial addition result is as expected |

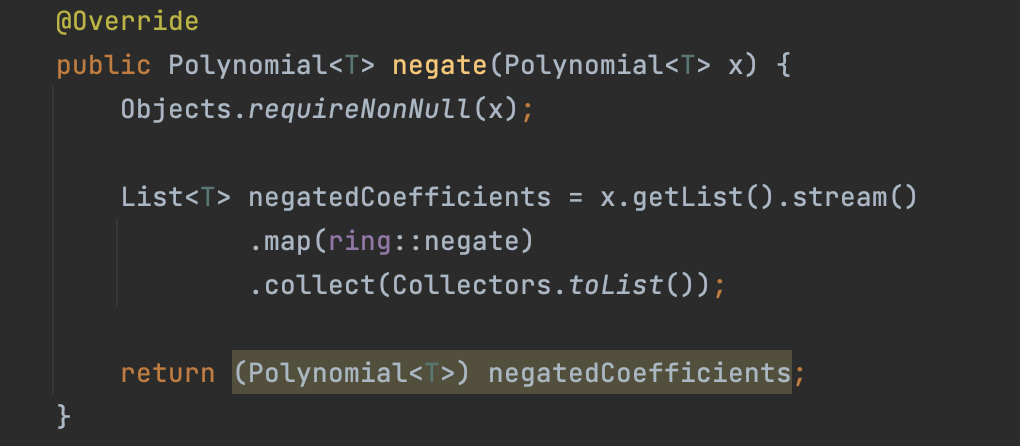
**Stress Test:**

Test condition: 1,000 elements in points of data.

Tested by iterating through the data values, plugged in points to the polynomial, and compared the assertion with 0.

**Tested Class:** Rings

**Method: negate(Polynomial<T> x)**

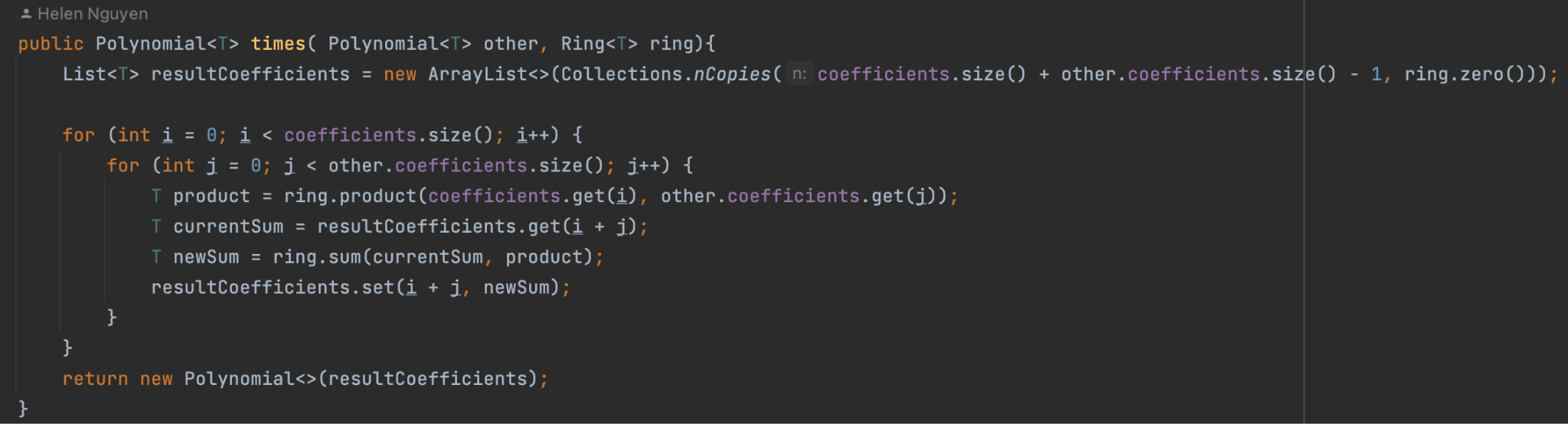


**Test Conditions:**

| Identifier | Goal | Notes | Condition |
| --- | --- | --- | --- |
| CC1 | Code Coverage |  | dataPoints is not empty |
| BD1 | Bad Data |  | dataPoints is empty |

**Test:**

| Test No. | Test | Test Condition | Conditions Satisfied | Assertion |
| --- | --- | --- | --- | --- |
| 1 | testNegate | CC1 | Valid input | Result is the negation of the input integer (e.g., -7) |
| 2 | testNegateNullInput | BD1 | - Null input | Expects a NullPointerException to be thrown |

**Method: times( Polynomial<T> other, Ring<T> ring)**

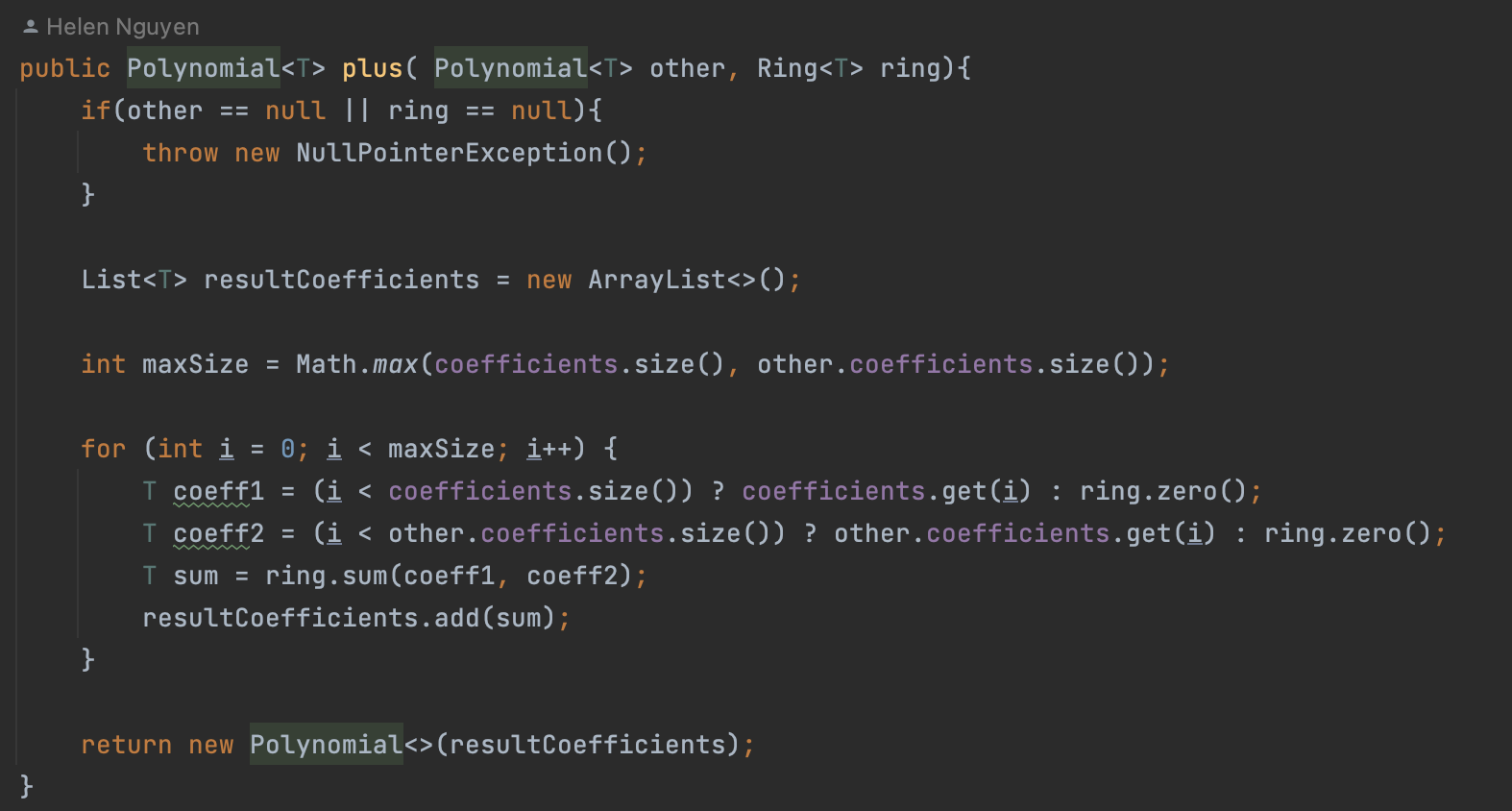
## **Test Conditions:**

| Identifier | Goal | Notes | Condition |
| --- | --- | --- | --- |
| CC1 | Code Coverage |  | Polynomial and ring is not empty |
| BD1 | Bad Data |  | Polynomial or ring is empty |
| B1 | Boundary |  | One or both polynomials are the zero polynomial |
| B2 | Boundary |  | Polynomial is the identity and the other is normal |
| B3 | Boundary |  | Coefficients contains identity |

## **Test:**

| Test No. | Test | Test Condition | Conditions Satisfied | Assertion |
| --- | --- | --- | --- | --- |
| 1 | testProduct | CC1 | - Valid input | Result is the product of 3 and 4 (e.g., 12) |
| 2 | testProductGeneric | CC1, B2,B3 | - Generic case | Result is an Optional containing the expected value (e.g., 2.0) |
| 3 | testProductZero | B1 | - Testing 0 | Result is an Optional containing the expected value (e.g., 0.0) |
| 4 | testProductNegative | CC1 | - Testing negative numbers | Result is an Optional containing the expected value (e.g., 6.0) |
| 5 | testProductNullY | BD1 | - Null Y input | Expects a NullPointerException to be thrown |
| 6 | testProductNullXY | BD1 | - Null X and Y inputs | Expects a NullPointerException to be thrown |
| 7 | testProductIntNullY | BD1 | - Null Y input (int version) | Expects a NullPointerException to be thrown |
| 8 | testProductIntNullXY | BD1 | - Null X and Y inputs (int version) | Expects a NullPointerException to be thrown |

**Method: plus( Polynomial<T> other, Ring<T> ring)**



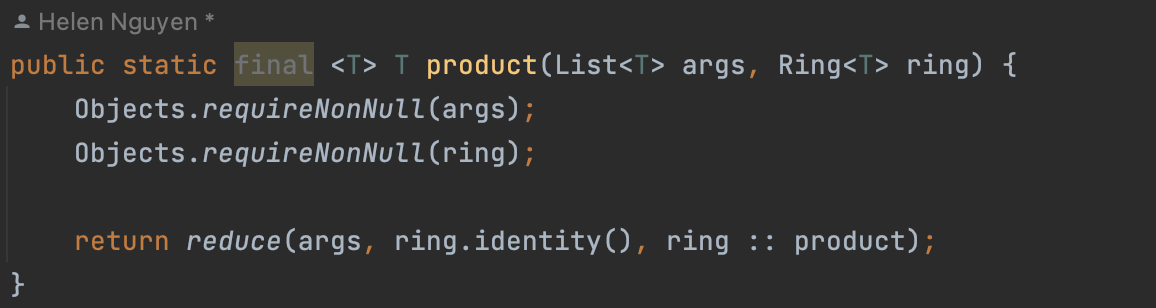
## **Test Conditions:**

| Identifier | Goal | Notes | Condition |
| --- | --- | --- | --- |
| CC1 | Code Coverage |  | Polynomial and ring is not empty |
| BD1 | Bad Data |  | Polynomial or ring is empty |
| B1 | Boundary |  | One or both polynomials are the zero polynomial |
| B2 | Boundary |  | Polynomial is the identity and the other is normal |
| B3 | Boundary |  | Coefficients contains identity |

## **Test:**

| Test No. | Test | Test Condition | Conditions Satisfied | Assertion |
| --- | --- | --- | --- | --- |
| 1 | testPlusSameSize | CC1 | - Same-sized Polynomials | Result is the expected Polynomial after addition |
| 2 | testPlusDifferentSize |  | - Different-sized Polynomials | Result is the expected Polynomial after addition |
| 3 | testPlusEmptyAndNonEmpty |  | - One Polynomial is empty | Result is the non-empty Polynomial |
| 4 | testPlusNull |  | - Both Polynomials are null | Expects a NullPointerException to be thrown |
| Note: There are also tests done for different data types but this report reflects Integer and Double since reporting all would be redundant. | | | | |

**Method: product(List<T> args, Ring<T> ring)**

****

## **Test Conditions:**

| Identifier | Goal | Notes | Condition |
| --- | --- | --- | --- |
| CC1 | Code Coverage |  | Polynomial and ring is not empty |
| BD1 | Bad Data |  | Polynomial or ring is empty |
| B1 | Boundary |  | One or both polynomials are the zero polynomial |
| B2 | Boundary |  | Polynomial is the identity and the other is normal |
| B3 | Boundary |  | Coefficients contains identity |

**Test:**

| Test No. | Test | Test Condition | Conditions Satisfied | Assertion |
| --- | --- | --- | --- | --- |
| 1 | testProductDoubleGeneric | CC1, B2, B3 | - Generic case | Result is an Optional containing the expected value (e.g., 2.0) |
| 2 | testProductDoubleZero | CC1, B1 | - Testing 0 | Result is an Optional containing the expected value (e.g., 0.0) |
| 3 | testProductDoubleNegative | CC1, B2, B3 | - Testing negative numbers | Result is an Optional containing the expected value (e.g., 6.0) |
| 4 | testProductDoubleNullY | CC1, BD1 | - Null Y input | Expects a NullPointerException to be thrown |
| 5 | testProductDoubleNullXY | BD1 | - Null X and Y inputs | Expects a NullPointerException to be thrown |
| 6 | testProductIntegerValid | CC1 | - Valid input | Result is the expected product (e.g., 12) |
| 7 | testProductIntegerNullY | BD1 | - Null Y input | Expects a NullPointerException to be thrown |
| 8 | testProductIntegerNullXY | BD1 | - Null X and Y inputs | Expects a NullPointerException to be thrown |