COMP 250 - Lecture 7

Polynomials Data + Methods
coefficients accessor/mutator
constructors operations

public class Poly {
  // name of the class
double[] coefficients; // visible in Poly, other classes
  // in same package, classes that
  extend Poly

  // p + q → p.add(q) call to a library method
  public Poly add(Poly q) {
    // coefficients, length
    // make a result
    Poly res = new Poly (Math.max(p.getDegree(), q.getDegree()),
    p, q); // method we wrote last time

    In general, function calls are somewhat
    preferred to direct data access
    Not mandatory
    this. getDegree() the current object
```java
int d1 = this.getDegree();
int d2 = g.getDegree();
int d = Math.max(d1, d2);
Poly res = new Poly(d);

// Method calls can be nested;
// arguments are evaluated before method called (inside ➔ out)
```

Go over the two polynomials, add coefficients and copy left over.

```java
for (int i = 0; i < Math.min(this.getDegree(), g.getDegree()); i++) {
    res.coefficients[i] = this.coefficients[i] + g.coefficients[i];
}
```

Uses data directly.

```java
res.setCoefficients(i, getCoefficients(i) + g.getCoefficients(i));
```

One more for loop to copy.
```java
if (getDegree() < g.getDegree()) {
    // copy over from g -> for loop
    for (int i = 0; i < System.math.max(...);
        i++)
        res.coefficients[i] = g.coefficients(System.math.min(...));
}
```

```
// i = Math.min() -> Math.max(...)
```

"Big-oh": $O(n)$, $n$ is maximum degree
public class Test Poly

   // main

    Poly p = new Poly(4, 1, 2, 3); // 1 + 2x + 3x^2
    Poly q = new Poly(40, 4); // 4x
    Poly r = p.add(q); // add -> this

    System.out.println(p.eval(2));
    double res = p.eval(1);
    // eval works on object on which it is called
Evaluate a polynomial

\[ p(x) = a_0 + a_1 x + \ldots + a_m x^m \]

\[ a_0 + a_1 x + a_2 x^2 + \ldots \]

Naive way: \( x^k \to \) add to the result \( a_k x^k \)

Smarter way: intermediate variable \( x^{k-1} \)

\[ x^{k-1} x \rightarrow x^k \]

\[ \left( (a_m x + a_{m-1}) x + a_{m-2} \right) x + a_{m-3} \ldots + a_0 \]

coefficients: \( a_0, a_m \)
public double eval (double x) {
    double res = 0;
    for (int i = 0; i < coefficients.length; i++) {
        res = res + coefficients[coefficients.length - i - 1] * x;
        res = res + coefficients[coefficients.length - i - 1] * x;
    }
    // Make sure indices are ok!
    // could make a down-going loop
    return res;
}

Primitive equations \rightarrow constant time
for: 1 less than length of array of coefficients.
O(n) n-degrees of the polynomial
Inheritance

Poly - any polynomial
- special cases: quadratic; positive coeffs

public class PositivePoly extends Poly {
    // Positive Poly will have all data & methods of
    // Poly + more stuff (specialized)
    public PositivePoly() {
        super();
        name of class
    }
    // call something from parent class (constructor)