COMP 250 - lecture 7

Polynomials   Data + Methods
coefficients   accessor/mutator
constructors
operations

public class Poly {
everyone can use it   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 ($\boxed{\text{Math.max}}$(getDegree(),
q.getDegree()));
method we wrote last time

In general, function calls are somewhat preferred to direct data access
Not mandatory.

this.getDegree()
the current object

L7-2

```
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 coefs and copy left over.

for loop. → min degree

```
for (int i=0; i < Math.min(getDegree(), g.getDegree());
                    i++) {
```

~~for (int j=0; j<Math.min(. .); j++)~~

```
res.coefficients[i] = coefficients[i] +
                      g.coefficients[i];
```

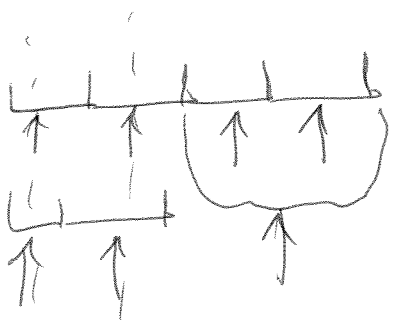
uses data directly.

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

One more for loop to copy

L7-3

```
if (getDegree() < q.getDegree()) {
  // copy over from q -> for loop
  for (int i = 0; i < Math.max(...) -
                       - Math.min(); i++)
     res.coefficients[i] = q.coefficients(Math.min(...)
                                           + i)
} // i = Math.min() -> Math.max(...)
```



i = 0 ... Max - min

i = min ... Max

"Big-oh" : O(n), n is maximum degree

(L7-4)

public class TestPoly {

... main ... {

Poly p = new Poly({1, 2, 3}); // 1+2x+3x²

Poly q = new Poly({0, 4}); // 4x

Poly r = p.add(q);

$\rightsquigarrow$ add → **this**

Syst.out.print(p.eval(2));

double res = p.eval();

↑ eval works on object on which it is called

Evaluate a polynomial

L7-5

$$P(x) = a_0 + a_1 x \dots + a_n x^n$$

$$a_0 + a_1 x + a_2 x^2 \dots$$

Naive way: $x^k \rightarrow$ add to the result $a_k x^k$

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

$$x^{k-1} x \rightarrow x^k$$

$$(((\underline{a_n} x + a_{n-1}) x + a_{n-2}) x + a_{n-3}) \dots + a_0$$

coefficients

L7-6

public double eval (double x){

double res = 0;

for (int i = ~~0~~ 1; i < coefficients.length; i++){

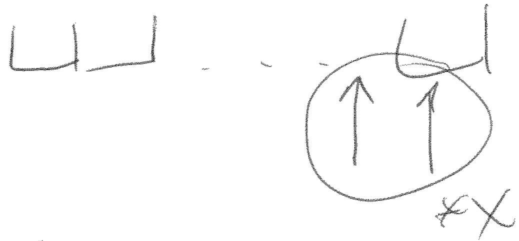
res = res + (coefficients[coefficients.length - i]*x + coefficients[coefficients.length - i - 1]);

} // Make sure indices are ok!

//could make a down-going loop

return res;

}



Primitive operations → constant time

for: 1 less than length of array of coef.

$O(n)$ n - degree of the polynomial

Inheritance

Poly – any polynomial

– special cases: quadratic; positive coefs

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