

Polynomial - specification

```
class Poly {
// Overview: Polys are immutable polynomials
// with integer coefficients. A typical Poly
// is  $c_0 + c_1x + c_2x^2 + \dots$ 
public Poly()
    // effects: makes a new Poly = 0
public Poly(int c, int e)
    throws NegExponentExn //when e < 0
    // effects: makes a new Poly =  $cx^e$ 

public int degree()
    // returns: the degree of this,
    // i.e. the largest exponent with a non-zero coefficient.
    // note: Returns 0 if this = 0.
public int coeff(int e)
    throws NEE //when e < 0
    // returns: the coefficient of
    // the term of this whose exponent is e

public Poly add(Poly q)
    // returns: the Poly = this + q
public Poly mul(Poly q)
    // returns: the Poly = this * q
public Poly minus()
    // returns: the Poly = -this
}
```

431 S'06 © ABorgida

3

Poly - specification

```
public Poly clone()
public boolean equals(Poly q)
public String toString()

public Iterator nonzeros()
    // returns generator producing only those
    // exponents whose coef is not zero
} // end of Poly

public interface Iterator { //spec'd by Java!
    public boolean hasNext();
    public Object next() throws NoSuchElementException;
    public void remove() throws IllegalStateException;
}
```

431 S'06 © ABorgida

4

Abstract Datatype + Iterator Module

example

431 S'06 © ABorgida

1

Polynomial module

- Based on Liskov&Gutttag
 1. Specification
 - mutable vs immutable type
 2. Use
 3. Implementation

431 S'06 © ABorgida

2

Poly - implement

```
public class Poly{
private int[] trms;
private int deg;
public Poly(int c, int e){
    if (e < 0) throw NEE;
    if (c==0) {trms=new int[1];deg=0; return;}
    trms = new int[e+1];
    for(int j=0; j<e;j++) trms[j]=0;
    trms[e]=c;
    deg= e;
}

public Poly mul (Poly q)
    throws NPE //when q is null
    { //Effects: returns the Poly this * q
    if ((q.deg ==0 && q.trms[0] == 0) ||
        (deg == 0 && trms[0] ==0))
        return new Poly();
    /* note:since immutable, cannot modify 'this'*/
    Poly r = new Poly (deg+q.deg);
    r.trms[deg+q.deg]=0;
    for(int i=0; i<=deg; i++)
        for(int j=0; j<+q.deg;j++)
            r.trms[i+j] += trms[i]*q.trms[j];
    return r; }
}
```

431 S'06 © ABorgida

7

Poly - alternate implement

The previous implementation wastes a lot of space for "sparse" polynomials -- ones with high degrees but few terms

```
public class Poly{
//implemented as a linked list of cells, each with fields for Coef
and Exponent:

private class Term{
    int coef;
    int expon;
    Term next;
    Term(c,e){coef=c;expon=e;next=null; return;}

private Term terms;
private int deg;
public Poly(int c, int e){
    if (e < 0) throw NEE;
    if (c==0) e=0;
    trms = new Term(c,e);
    deg= e;
}

//Poly.add, etc. chain the terms into lists
```

431 S'06 © ABorgida

8

Poly - use

```
Poly q = new Ploy(2,3); //2 x^3
Poly q = q.add(new Poly(4,6)); // 2x^3 + 4x^6
int e = q.degree(); // 6
int c = q.coef(5); // 0
int c = q.coef(3); // 2

//this prints all terms from 0 to 6
for (e = 0; e<= q.degree; e++){
    print q.coef(e) + 'x^' + e;
}

//this will only print the non-zero terms
Iterator qit = q.nonzeros();
while ( qit.hasNext() ){
    e = (Integer) qit.next();
    c = q.coef(e);
    print c + 'x^' + e;
}
```

431 S'06 © ABorgida

5

Poly - other things in a module

```
public class Poly{
// could export constants, which are used frequently
public static final Poly zero = new Poly();
public static final Poly one = new Poly(1,1);

/* sample use
    if ( Poly.zero.equal(q) ) ...
or if we are careful to always reduce the zero poly to a canonical
form, then
    if ( Poly.zero == q) ...
will also work, and will be very efficient
*/

// in Java 1.5, enumerated types
public enum PolyKinds {prime,factorable}
```

431 S'06 © ABorgida

6

e.g., Managing a list of names --2

procedures:

initialize()

modifies: the list

effect: sets up an empty person list

```
boolean addPerson(   NameString firstN IN,
                    NameString lastN IN,
                    Errors msg OUT)
```

requires: initialize() has been called already

modifies: 'this' - the list itself

effect: creates a person with this name and adds it to the list;

returns: true iff successful

signals: if name already there, msg= ...

if there is no more room, msg= ...

```
boolean removePerson(NameString firstN IN, NameString lastN IN)
```

...

*Need to export or
import
type Errors*

e.g., Managing a list of names -- 3

procedures:

- In languages where there is no automatic garbage collection, there should be procedures for de-allocating space and cleaning up:
 - destroyPersonList()
 - terminateGenerator(PersonIterator pi)
- If we wanted to have several person lists, in non-object oriented languages, we would need to pass the list in as an extra argument to all procedures on page 2

Poly - implement iterator

```
public Iterator nonzeros() {
    return new PolyGen(this);
}

// private inner class, inside Poly
private static class PolyGen implements Iterator {
    private Poly p; // the Poly being iterated
    private int n; // the next term to consider

    PolyGen(Poly over){
        p = over;
        //n = 0; this will be incorrect for the zero poly
        if(p.trms[0] == 0) n=1; else n=0;
    }

    public boolean hasNext()
    {return n<= p.deg;}
    public Object next () throws NSEE{
        for(int e = n; e <= p.deg; e++) {
            if (p.trms[e] != 0) {
                n= e+1;
                return new Integer(e);
            }
        }
        throw new NSEE("Poly.terms");
    } //end next()
} // end PolyGen
} // Poly
```

e.g.2, Managing a list of persons

//in non-object oriented style

MODULE PersonList

Description: Manages a list of persons (each consisting of a first and last name). Names can be added or removed from the list (mutable type), and the list can be traversed in alphabetical order of first name or last name.

Secrets: How names are stored (in order to save space), and the way in which sorted lists are produced (on demand or pre-sorted)

Imports: type NameString with function int compare(NS , NS)

Exports:

type Person with procedures:

```
NameString getFirstName(Person p IN)
```

returns first name of the person p

```
NameString getLastName(Person p IN)
```

returns last name of the person p

requires: arg != null
but we won't say this

e.g., Managing a list of names -- Java version

public class PersonList{ //more methods

```
Iterator getFirstNameGenerator()
    returns: a generator where persons are returned in increasing
            alphabetical order by first name;
Iterator getLastNameGenerator()
    returns: a generator where persons are returned in increasing
            alphabetical order by first name;
}
```

package PersonManager

```
public interface Person{ //this will be the kind of object returned by the
    generator's next() function, so we must export this type too!
    public
        NameString getFirstName();
        NameString getLastName();
}
```

Note that we do not want to allow people outside to construct objects of this type!

e.g., Managing a list of names -- Java version

//Sample use

```
import PersonManager; //which in turn imports XXX, which has NameString
PersonList pl = new PersonList();
pl.add( new NameString('Fred'), new NameString('Hoyle'));
...
Iterator byFirst = pl.getFirstNameGenerator()
Person p;
while (byFirst.hasNext()) {
    p = (Person) byFirst.next();
    Output '(' + p.getFirstName() + ' ' + p.getLastName() + '\n';
}
...
```

e.g., Managing a list of names --4

For iterating through the persons:

Exports:

type PersonIterator with procedures

```
boolean hasNext(PersonIterator pi IN)
    returns: true iff there is at least one more name to go thru in pi
```

```
Person next(PersonIterator pi IN)
```

requires: hasNext(pi) returns true ← *Note how this is different from Java iterator, where you can call next() even when hasNext() was false*
returns: next person and advance cursor in Generator

```
PersonIterator getFirstNameGenerator()
    requires: initialize() has been called;
    returns: a generator where persons are encountered in increasing
            alphabetical order by first name; cursor set before the first person
```

```
PersonIterator getLastNameGenerator()
```

...

e.g., Managing a list of names -- Java version

package PersonManager;

public class PersonList{

//Overview:...

//Imports: class NameString from some other package XXX;

// NameString assumed to extend java.lang.Comparable

import XXX;

public

PersonList()

effect: sets up an empty person list

void addPerson(NameString firstN, NameString lastN)

throws DuplicateXn //when name pair is already there

NoRoomXn //when there is no more room in the list

modifies: this

effect: creates a person with this name and adds it to the list;

void removePerson(NameString firstN, NameString lastN)

throws MissingXn //when the person is not in the list

modifies: this

effect: removes the person with the specified name

}

public class DuplicateXn extends RuntimeException{

public DuplicateXn(super());}

Because Java is call by value, everything is IN; but can be modified, if mutable!

e.g., Managing a list of names -- implement iterator

```
//inside PersonList{
int size; // length of list
NameString[] first, NameString[] last; //array of first and last names
int[] firstOrder, int[] lastOrder; // array of ordered indeces

private class MyPerson implements Person{
private int j;
MyPerson(int index){j=index;}
public NameString getFirstName(){return FirstNames[j];}
} //MyPerson
private class NameGen implements Iterator{
private PersonList pl;
private int index;

NameGen(PersonList it){pl = it; index=0;}
public boolean hasNext(){return (index <= pl.size);}
public Person next()
{if ( index <= pl.size ) return new Person(index++); else throw
NSVE;}
}
public Iterator getFirstNameGenerator() {return new NameGen(this,0);}
```

e.g., Managing a list of names -- Java (3)

• What is bad about Java from SE point of view:

- classes expose private part/implementation, even if you cannot access them -- would be nice to be able to read only a “header” which shows the public material -- the ‘specs’
- interfaces (which would be one natural mechanism for the above) only allow method names, not nested classes, constants,...
- “package” is the only mechanism for grouping several classes, exceptions, etc. into one conceptual ‘module’; this is weak

e.g., Managing a list of names -- Java 1.5

```
Iterator<Person> getFirstNameGenerator()
    returns: a generator where persons are returned in increasing
    alphabetical order by first name; cursor set before the first
    person
//Smample use
import NameManager;
PersonList pl = new PersonList();
pl.add( new NameString('Fred'), new NameString('Hoyle'));
...
Iterator <Person> y = pl.getFirstNameGenerator()
Person p;
while (y.hasNext()) {
    p = y.next();
    Output '(' + p.getFirstName() + ' ' + p.getLastName() + '\n';
}
y = pl.getLastNameGenerator()
while (y.hasNext()) {
    p = y.next();
    Output '(' + p.getFirstName() + ' ' + p.getLastName() + '\n';
}
```

e.g., Managing a list of names -- implement

```
//inside PersonList{
int size; // length of list
NameString[] first, NameString[] last; //array of first and last names
int[] firstOrder, int[] lastOrder; // array of ordered indeces into First
and Last respectively;
/* e.g., first[firstOrder[size]],last[firstOrder[size]] is the full name of the
person whose first name is alphabetically last */

void add(NameString f, NameString l){
size++;
first[size]= f;
last[size]=l;
// put value size into firstOrder so first names sorted
// put value size into lastOrder ...
}
```