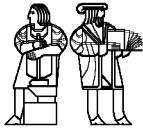


# 6.170 Lecture 9 Abstract Data Types



John Guttag  
MIT EECS

## Outline

1. What is an ADT?
2. How to specify an ADT
  - immutable
  - mutable
3. Things to know about the ADT methodology

©Michael Ernst/John Guttag

Spring 2001

2

## What Is an ADT?

### Procedural abstraction

Abstracts from the details of procedures  
A specification mechanism

### Data abstraction (Abstract Data Type, or ADT):

Abstracts from the details of data representation  
A specification mechanism  
+ a way of thinking about programs and designs

©Michael Ernst/John Guttag

Spring 2001

3

## Why We Need Abstract Data Types

### Programming is not usually about

Inventing and describing algorithms

### It is more often about

Organizing and manipulating data

### Leads designers to start by

Designing data structures  
Writing code to access and manipulate data

### Problematical because

Decisions about data structures made too early  
Duplication of effort in creating derived data  
Very hard to change key data structures

©Michael Ernst/John Guttag

Spring 2001

4

## What Is an ADT, revisited

Abstract from organization to meaning of data

Abstract from structure to use

### Avoid concern with

right\_triangle = struct [base, altitude: real]  
vs.  
right\_triangle = struct [base, hypot, angle: real]

### Instead think of type as a set of operations

E.g., create, base, altitude, bottom\_angle, ...

Force users to call operations to access data

©Michael Ernst/John Guttag

Spring 2001

5

## Are These Classes the Same or Different?

```
class Point {                class Point {
  public real x;              public real r;
  public real y;              public real theta;
}                               }
```

**Different:** can't replace one with the other

**Same:** both classes implement the concept "2-d point"

### Goal of ADT methodology

Express the sameness  
Clients depend only on the concept "2-d point"

### Good because:

Performance optimizations  
Fix bugs  
Delay decisions

©Michael Ernst/John Guttag

Spring 2001

6

**Concept of 2-d point, as ADT**

```

class Point {
  // A 2-d point exists somewhere in the plane, ...
  public real x();
  public real y();
  public real r();
  public real theta();
  // can be created.
  public Point(); // new point at (0,0)

  // ... can be moved, ...
  public void translate(real delta_x,
                       real delta_y);
  public void scale_rot(real delta_r,
                       real delta_theta);
}

```

©Michael Ernst/John Guttag Spring 2001 7

**Abstract data type = objects + operations**

**Implementation hidden**  
**No operations on objects of the type except those provided by the abstraction**

©Michael Ernst/John Guttag Spring 2001 8

**How to Specify an ADT**

immutable	mutable
<pre> class typename {   1. overview   2. creators   3. observers   4. producers } </pre>	<pre> class typename {   1. overview   2. creators   3. observers   4. mutators } </pre>

©Michael Ernst/John Guttag Spring 2001 9

**Primitive Data Types Are ADTs**

**int is an immutable ADT:**

creators: 1, 2, ...  
 producers: + - \* / ...  
 observer: Integer.toString(int)

©Michael Ernst/John Guttag Spring 2001 10

**Poly: overview and creators**

```

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 n)
  // effects: makes a new Poly =  $cx^n$ , unless
  // throws: NegExponent when  $n < 0$ 
}

```


©Michael Ernst/John Guttag Spring 2001 11

**Notes on Overview and Creators**

**Overview**  
 Always say if mutable or immutable  
 Define abstract model for use in specs of ops  
 Difficult and vital!  
 Appeal to math if appropriate  
 Give example (reuse in operation definitions)

**Creators**  
 New object, not part of prestate: in *effects*, not *modifies*  
 Overloading: distinguish procs of same name by arglist  
 Example: Poly(int,int) creator declared to return  $cx^n$   
 Key feature of all ADTs, state in specs is abstract

©Michael Ernst/John Guttag Spring 2001 12

 Poly: observers

---


```

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 d)
    // returns: the coefficient of
    // the term of this whose exponent is d

```

©Michael Ernst/John Guttag Spring 2001 13

 Notes on Observers

---

**Observers**  
Used to obtain information about objects of the type  
Return values of other types  
Never modify the object  
Specification uses the abstraction from the overview


**this**  
The particular Poly object being worked on  
I.e., the target of the invocation

```

Poly x = new Poly(4, 3);
int c = x.coeff(3);
System.out.println(c); // prints 4

```

©Michael Ernst/John Guttag Spring 2001 14

 Poly: producers

---

```

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

} // end Poly

```


©Michael Ernst/John Guttag Spring 2001 15

 Notes on Producers

---

**Producers**  
Operations on a type that create other objects of the type  
Common in immutable types, e.g., *java.lang.String*  
String substring(int offs, int len)

©Michael Ernst/John Guttag Spring 2001 16

 IntSet: overview and creators

---


```

class IntSet {
    // Overview: IntSets are mutable, unbounded
    // sets of integers. A typical IntSet is
    // { x1, ..., xn }.

    public IntSet()
        // effects: makes a new IntSet = {}

```

©Michael Ernst/John Guttag Spring 2001 17

 IntSet: observers

---

```


public boolean isIn(int x)
    // returns: true if x ∈ this
    // else returns false

public int size()
    // returns: the cardinality of this

public int choose()
    // returns: some element of this
    // throws: EmptyException when size() == 0

```

©Michael Ernst/John Guttag Spring 2001 18

 **IntSet: mutators**

---

```


public void insert(int x)
    // modifies: this
    // effects:  this_post = this ∪ {x}

public void remove(int x)
    // modifies: this
    // effects:  this_post = this - {x}

} // end IntSet

```

©Michael Ernst/John Guttag      Spring 2001      19

 **Notes on Mutators**


---

**This is how we get all nonempty IntSets**

**Mutators**  
 Operations that modify an element of the type  
 Almost never modify anything other than *this*  
 Mutable ADTs may have producers too, but less common

**Must list this in modifies clause (if appropriate)**

©Michael Ernst/John Guttag      Spring 2001      20

 **Exposing the Rep**

---

```


Point p1 = new Point();
Point p2 = new Point();
Line line = new Line(p1,p2);
p1.translate(5, 10); // move point p1

```

**Is Line mutable or immutable?**  
**Implementation dependent!**  
 If Line creates an internal copy: immutable  
 If Line stores a reference to p1,p2: mutable

**Lesson: storing a mutable object in an immutable collection can expose the representation**

©Michael Ernst/John Guttag      Spring 2001      21

 **ADTs and Java Language Features**

---

**Java classes**  
 Make operations in the ADT public  
 Make other ops and fields of the class private  
 Clients can only access ADT operations      good  
 Clients can/must read implementation      bad

**Java interfaces**  
 Clients only see the ADT, not the implementation      GOOD  
 Allow multiple implementations in same program      good  
 Cannot include creators      BAD

**My suggestion**  
 Write and rely upon careful specifications  
 Uses classes

©Michael Ernst/John Guttag      Spring 2001      22