CS3500: Object-Oriented Design
Spring 2014

Class 7
1.28.2014

Northeastern University
College *of* Computer and Information Science

Today...

- Liskov - Chapter 5: Data Abstraction
- Iterators
- Queue
- Understanding an implementation

Assignment 4

- Implement MyMap
- Due: Friday, January 31, 2014 at 11:59 pm

Review of Liskov Chapters 1-4

Abstraction Mechanisms

- Abstraction by parameterization
- Abstraction by specification

Kinds of Abstraction

- Procedural abstraction
- Data abstraction
- Iteration abstraction

Chapter 5: Data Abstraction

[Liskov]

What is data abstraction?

What is data abstraction?

A type of abstraction that allows us to introduce new types of data objects.

What must we define with a new data type?

What must we define with a new data type?

- set of objects
- set of operations characterizing the behavior of the objects

```
data abstraction = <objects, operations>
```

Abstract Data Type (ADT)

Review

- What is an ADT?
  - set of data
  - set of operations
  - description of what operations do
- Within this course, when discuss ADTs, we will discuss them using:
  - a signature: names of operations and types
  - a specification: agreement between client and implementors

Objects

- Object
  - a programming entity that contains state (data) and behavior (methods)
- Objects we've discussed so far…
  - `String`
  - `Point`
  - `Scanner`
  - `Random`
  - `File`
  - arrays

Objects

- **State**: a set of values (internal data) stored in an object

- **Behavior**: a set of actions an object can perform, often reporting or modifying its internal state

Client Code

- Objects themselves are not complete programs; they are components that are given distinct roles and responsibilities

- Objects can be used as part of larger programs to solve programs

- **Client (or Client Code)**: code that interacts with a class or objects of that class

What do we gain from data abstraction?

Abstraction Barrier

- Every piece of software has, or should have, an abstraction barrier that divides the world into two parts: clients and implementors.
  - The clients are those who use the software. They do not need to know how the software works.
  - The implementors are those who build it. They need to know how the software works.

Abstraction Barrier

- Client
  - Knows the behavior of the data type
  - Doesn't know how the data type was implemented, but can use the data type based on

    the specs

Abstraction Barrier

Implementor

- Knows the behavior of the data type
- Knows how the data type was implemented

Which abstraction mechanisms are used with data abstraction?

Which abstraction mechanisms are used with data abstraction?

- Abstraction by parameterization
- Abstraction by specification

Specifications

- Formal
- Informal

```
visibility class dname{
  //OVERVIEW: A brief description of the
  // behavior of the type's objects goes
  // here.

  //constructors
  //specs for constructors go here

  //methods
  //specs for methods go here
}
```

From [Liskov]

```
public class IntSet{
  //OVERVIEW: IntSets are mutable, unbounded
  //   sets of integers.
  //   A typical IntSet is {x1,...,Xn}

  //constructors
  public IntSet()
    //EFFECTS: Initializes this to be empty

  //methods
  public void insert (int x)
    //MODIFIES: this
    //EFFECTS: Adds x to the elements of
    //   this, i.e.,
    //   this_post = this + {x}.

  public void remove (int x)
    //MODIFIES: this
    //EFFECTS: Removes x from this, i.e.,
    // this_post = this - {x}

  public boolean isIn (int x)
    //EFFECTS: If x is in this returns true
    //else returns false

  public int size ()
    //EFFECTS: Returns the cardinality of
    //this

  public int choose () throws Empty Exception
    //EFFECTS: If this is empty, throws
    //   EmptyException else
    //   returns an arbitrary element of this
}
```

From [Liskov]

```
public class Poly{
  // OVERVIEW: Polys are immutable polynomials with integer coefficients
  //    A typical Poly is c_0 + c_1 x + ...

  //constructors
  public Poly()
    //EFFECTS: Initializes this to be the zero polynomial

  public Poly(int c, int n) throws NegativeExponentException
    //EFFECTS: If n<0 throws NegativeExponentException else
    //  initializes this to be the Poly cx^n

  //methods
  public int degree()
    //EFFECTS: Returns the degree of this, i.e., the largest exponent
    //  with a non-zero coefficient. Returns 0 if this is the zero Poly.

  public int coeff(int d)
    //EFFECTS: Returns the coefficient of the term of this whose exponent is d

  public Poly add(Poly q) throws NullPointerException
    //EFFECTS: If q is null throws NullPointerException else
    //  returns the Poly this + q.

  public Poly mul(Poly q) throws NullPointerException
    //EFFECTS: If q is null throws NullPointerException else
    //  returns the Poly this * q.

  public Poly sub(Poly q) throws NullPointerException
    //EFFECTS: If q is null throws NullPointerException else
    //  returns the Poly this -q.

  public Poly minus()
    //EFFECTS: Returns the Poly - this.
}
```

From [Liskov]

Implementing Data Abstractions

Access in Implementation

Access Modifiers

- `private` - accessible only within the same class
- (default) - accessible only within the same package
- `protected` - accessible within the same package and also accessible within subclasses
- `public` - accessible everywhere

Effective Java
[Bloch]

- Item 13: Minimize the accessibility of classes and members
- Item 45: Minimize the scope of local variables
- Item 14: In public classes, use accessor methods, not public fields

Records

Sidebar 5.1 - `equals`, `clone`, and `toString`

[Liskov, p.94]

- Two objects are `equals` if they are behaviorally equivalent. Mutable objects are `equals` only if they are the same object; such types can inherit `equals` from `Object`. Immutable objects are `equals` if they have the same state; immutable types must implement `equals` themselves.
- `clone` should return an object that has the same state as its object. Immutable types can inherit `clone` from `Object`, but mutable types must implement it them selves.
- `toString` should return a string showing the type and current state of its object. All types must implement `toString` themselves

Item 8: Obey the general contract when overriding `equals`

[Bloch]

The equals method implements an equivalence relation. It is:

- Reflexive
- Symmetric
- Transitive
- Consistent
- For any non-null reference value x, x.equals(null) must return false.

Item 10: Always override `toString`

[Bloch]

Iterators

Related Readings

- Sestoft Section 13: Interfaces
- Sestoft Section 22.7: Going Through a Collection: Interfaces Iterator<T> and Iterable<T>
- Liskov: Chapter 6 - Iteration Abstraction

```
Scanner input = new Scanner(System.in);
int count = 0;
while (input.hasNext()) {
   String word = input.next();
   count++;
}
System.out.println("count = " + count);
```

Iterators

[Lewis & Chase]

- An *iterator* is an object that provides the means to iterate over a collection.
- Provide methods that allow the user to acquire and use each element in a collection in turn.

```
//An iterator over a collection.
public interface Iterator<E> {
  // Returns true if the iteration has more
  // elements.
  public boolean hasNext ();

  // Returns the next element in the iteration.
  public E next ();

  // Removes from the underlying collection the last
  // element returned by the iterator (optional).
  public void remove ();
}
```

while loop

```
IntIterator it = new IntIterator();

while(it.hasNext()){
    System.out.println(it.next());
}
```

for each statement

```
for (tx x : expression)
    body
```

for each statement

```
for (tx x : expression)
    body
```

Iteration Abstraction [Liskov, p.130]

- An *iterator* is a procedure that returns a *generator*. A data abstraction can have one or more iterator methods, and there can also be standalone iterators.
- A generator is an object that produces the elements used in the iteration. It has methods to get the next element and to determine where there are any more elements. The generator's type is a subtype of `Iterator`.
- The specification of an iterator defines the behavior of the generator; a generator has no specification of its own. The iterator specification often includes a requires clause at the end constraining the code that uses the generator.

Queue

- Similar to list
- First In, First Out (FIFO)

- Enqueue
- Dequeue