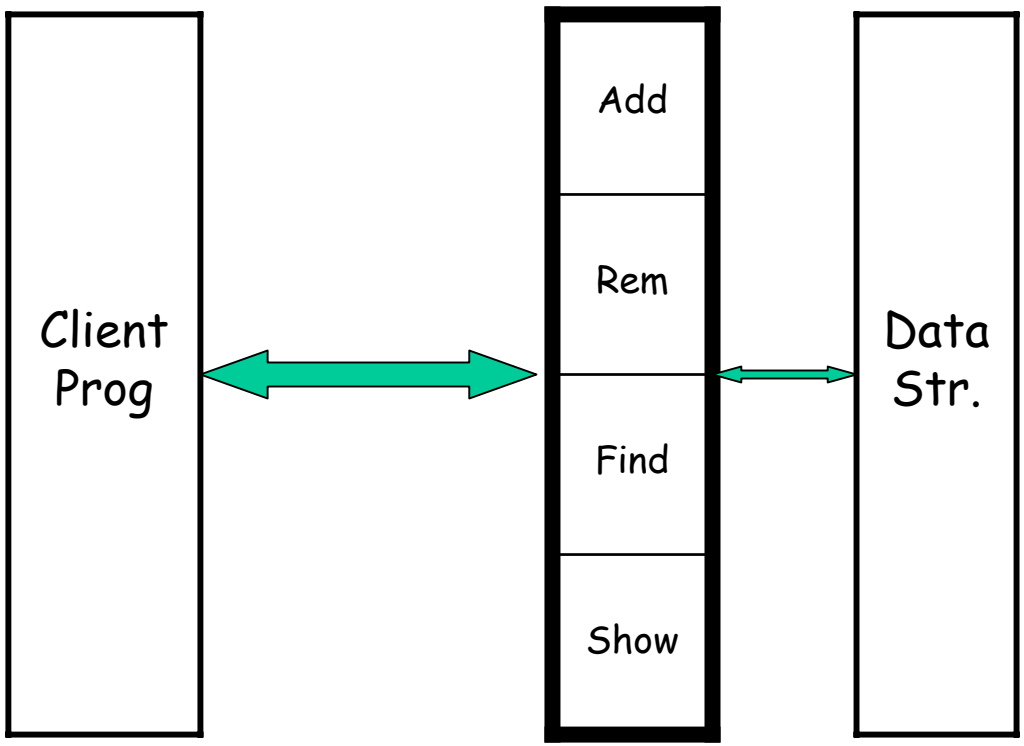
Notes on access restrictions

- A source code file MyClass.java is a compilation unit and can contain at most one public class. Furthermore, if there is a public class in that file, it must be called MyClass. Upon compilation, a .class file is created for each class.
- Creating a package implies a certain directory structure for each package, and the directory must be searchable using the CLASSPATH environmental variable.
- public, package access, protected, private: access hierarchy.
- A class (except inner classes) cannot be private/protected. But one could make all constructors of a class private.

```
public class BinarySearch // Fig 5.11, pg168
{
   public static final int NOT_FOUND = -1;

  public static int binarySearch
              ( Comparable [ ] a, Comparable x )
   {
      int low = 0;
      int high = a.length - 1;
      int mid;
      while( low <= high )
      {
         mid = ( low + high ) / 2;
         if( a[ mid ].compareTo( x ) < 0 )
            low = mid + 1;
         else if( a[ mid ].compareTo( x ) > 0 )
            high = mid - 1;
         else
            return mid;
      }
      return NOT_FOUND;     // NOT_FOUND = -1
   }

   // Test program
   public static void main( String [ ] args )
   {
      int SIZE = 8;
      Comparable [ ] a = new Integer [ SIZE ];
      for( int i = 0; i < SIZE; i++ )
         a[ i ] = new Integer( i * 2 );

      for( int i = 0; i < SIZE * 2; i++ )
         System.out.println( "Found " + i + " at " +
              binarySearch( a, new Integer( i ) ) );

   }
}
```

Abstract Data Types



Linear Lists

- It is an ordered collection of elements.
- Lists have items, size or length.
- Elements may have an index.
- Main operations:
  - isEmpty(), size(),
  - get(idx), indexOf(elem),
  - remove(idx), add(idx, elem),
  - display()
- Java's linear lists:
  - java.util.ArrayList and java.util.LinkedList.

Using Iterators

- Why use them?
- Compare these 2 pieces of code:
  - for (int j = 0; j < A.size(); j++)
    visit(A.get(j))
  - iterator h = A.iterator();
    while (h.hasNext())
    visit(h.next());
- Which one is better? Why?

```
package weiss.util;

public interface List
        extends Collection
{
    Object get( int idx );
    Object set( int idx,
            Object newVal );
    Iterator listIterator( int pos );
}
```

```
class TestArrayList
{
    public static void main( String [ ] args )
    {
        ArrayList lst = new ArrayList( );
        lst.add( "2" );  lst.add( "4" );
        ListIterator itr1 = lst.listIterator( 0 );
        System.out.print( "Forward: " );
        while( itr1.hasNext( ) )
            System.out.print( itr1.next( ) + " " );
        System.out.println( );
    }
}
```

```
// Fig 6.16,6.17, pg 201, 202
package weiss.util;

public interface List
        extends Collection
{
    Object get( int idx );
    Object set( int idx,
        Object newVal );
    ListIterator listIterator( int pos );
}

public interface ListIterator
        extends Iterator
{
    boolean hasPrevious( );
    Object previous( );
    void remove( );
}
```

```
class TestArrayList // Fig 6.18, pg 203
{
    public static void main( String [ ] args )
    {
        ArrayList lst = new ArrayList( );
        lst.add( "2" );  lst.add( "4" );
        ListIterator itr1 = lst.listIterator( 0 );
        System.out.print( "Forward: " );
        while( itr1.hasNext( ) )
            System.out.print( itr1.next( ) + " " );
        System.out.println( );

        System.out.print( "Backward: " );
        while( itr1.hasPrevious( ) )
            System.out.print( itr1.previous( ) + " " );
        System.out.println( );

        System.out.print( "Backward: " );
        ListIterator itr2 = lst.listIterator( lst.size( ) );
        while( itr2.hasPrevious( ) )
            System.out.print( itr2.previous( ) + " " );
        System.out.println( );
    }
}
```

```
// Fig 6.5-6.7, pg 189
package weiss.ds;

public class MyContainer
{
   private Object [ ] items;
   private int size = 0;

   public Object get( int idx )
   public boolean add( Object x )
   public Iterator iterator( )
   // Factory method: type of iterator is unknown.

   private class LocalIterator implements Iterator
   {
      private int current = 0;

      public boolean hasNext( )
      public Object next( )
   }
}
```

Caveats about iterators

- Consider, for e.g. the following problem: Delete all students that have dropped the class (have the drop flag ON) from the class roster.

```
Iterator itr = c.iterator();
while (itr.hasNext() && (dropped(itr))
        remove(itr);
```

- What item is "current" if it has been "removed".
- What happens if we are within a "for-loop"?
  - Removal might change for-loop bounds.

```
// pg 205
package weiss.util;

public class LinkedList extends AbstractCollection implements List
{
   public void addFirst( Object x )
   public void addLast( Object x )
   public Object getFirst( )
   public Object getLast( )
   public Object removeFirst( )
   public Object removeLast( )
}
```

```
public interface Stack
{
   public Object push( Object x );
   public Object pop( );
   public boolean isEmpty( );
}

public interface Queue
{
   public boolean isEmpty( );
   public void enqueue( Object x );
   public Object dequeue( );
}
```