Functors

- A functor is an object with no data and a single method.
- Functors can be passed as parameters.
- Since these classes are very “small”, they are usually implemented as a Nested Class wherever they are needed.
- Nested classes are defined inside other classes and it is essential that it be declared as “static”. If it is not declared as “static”, then it is an “inner” class (not nested).
- Nested classes act as members of the “outer” class, and can be declared as private, public, protected, or package visible.
- A nested class can access private fields and members of the “outer” class.
- Functors can also be implemented as a Local Class or as an Anonymous Class.

```
// Fig 4.29 & 4.30, pg 127

import java.util.Comparator;

class OrderRectByArea implements Comparator
{
    public int compare( Object obj1, Object obj2 )
    {
        SimpleRectangle r1 = (SimpleRectangle) obj1;
        SimpleRectangle r2 = (SimpleRectangle) obj2;

        return( r1.getWidth()*r1.getLength() -
                r2.getWidth()*r2.getLength() );
    }
}
```

Functors

```
public class CompareTest
{
    public static Object findMax( Object [ ] a,
                                  Comparator cmp )
    {
        int maxIndex = 0;
        for( int i = 1; i < a.length; i++ )
            if( cmp.compare( a[ i ], a[ maxIndex ] ) > 0 )
                maxIndex = i;
        return a[ maxIndex ];
    }

    public static void main( String [ ] args )
    {
        Object [ ] rects = new Object[ 4 ];
        rects[ 0 ] = new SimpleRectangle( 1, 10 );
        rects[ 1 ] = new SimpleRectangle( 20, 1 );
        rects[ 2 ] = new SimpleRectangle( 4, 6 );
        rects[ 3 ] = new SimpleRectangle( 5, 5 );

        System.out.println( "MAX WIDTH: " +
          findMax( rects, new OrderRectByWidth( ) ) );
        System.out.println( "MAX AREA: " +
          findMax( rects, new OrderRectByArea( ) ) );
    }
}
```

```
import java.util.Comparator;
// Fig 4.32 pg 130
class CompareTestInner1
{
   public static Object findMax( Object [ ] a,
                                             Comparator cmp )
   {
      int maxIndex = 0;
      for( int i = 1; i < a.length; i++ )
         if( cmp.compare( a[ i ], a[ maxIndex ] ) > 0 )
            maxIndex = i;

      return a[ maxIndex ];
   }

private static class OrderRectByArea
              implements Comparator
   {
      public int compare( Object obj1, Object obj2 )
      {
         SimpleRectangle r1 = (SimpleRectangle) obj1;
         SimpleRectangle r2 = (SimpleRectangle) obj2;

         return( r1.getWidth()*r1.getLength() -
               r2.getWidth()*r2.getLength() );
      }
   }
```

```
public static void main( String [ ] args )
   {
      Object [ ] rects = new Object[ 4 ];
      rects[ 0 ] = new SimpleRectangle( 1, 10 );
      rects[ 1 ] = new SimpleRectangle( 20, 1 );
      rects[ 2 ] = new SimpleRectangle( 4, 6 );
      rects[ 3 ] = new SimpleRectangle( 5, 5 );

      System.out.println( "MAX WIDTH: " +
         findMax( rects, new OrderRectByWidth( ) ) );
      System.out.println( "MAX AREA: " +
         findMax( rects, new OrderRectByArea( ) ) );
   }
}
```

Nested Classes

Local Classes

```
import java.util.Comparator;
// Fig 4.33 pg 131
class CompareTestInner2
{
   public static Object findMax( Object [ ] a,
              Comparator cmp )
   {
      int maxIndex = 0;
      for( int i = 1; i < a.length; i++ )
         if( cmp.compare( a[ i ], a[ maxIndex ] ) > 0 )
            maxIndex = i;

      return a[ maxIndex ];
   }
```

```
public static void main( String [ ] args )
   {  Object [ ] rects = new Object[ 4 ];
      rects[ 0 ] = new SimpleRectangle( 1, 10 );
      rects[ 1 ] = new SimpleRectangle( 20, 1 );
      rects[ 2 ] = new SimpleRectangle( 4, 6 );
      rects[ 3 ] = new SimpleRectangle( 5, 5 );

  // neither public nor static
  class OrderRectByArea implements Comparator
     {
        public int compare( Object obj1, Object obj2 )
        {
           SimpleRectangle r1 = (SimpleRectangle) obj1;
           SimpleRectangle r2 = (SimpleRectangle) obj2;

           return( r1.getWidth()*r1.getLength() -
                 r2.getWidth()*r2.getLength() );
        }
     }

   System.out.println( "MAX AREA: " +
            findMax( rects, new OrderRectByArea( ) ) );
   }
}
```

Anonymous Classes

```
class CompareTestInner3 // Fig 4.34, pg 132
{
public static void main( String [ ] args )
    {
        Object [ ] rects = new Object[ 4 ];
        rects[ 0 ] = new SimpleRectangle( 1, 10 );
        rects[ 1 ] = new SimpleRectangle( 20, 1 );
        rects[ 2 ] = new SimpleRectangle( 4, 6 );
        rects[ 3 ] = new SimpleRectangle( 5, 5 );
        System.out.println( "MAX WIDTH: " + findMax( rects, new Comparator( )
            {  // no name class, no constructor
                public int compare( Object obj1, Object obj2 )
                {
                    SimpleRectangle r1 = (SimpleRectangle) obj1;
                    SimpleRectangle r2 = (SimpleRectangle) obj2;
                    return( r1.getWidth() - r2.getWidth() );
                }
            }
        ) );
        System.out.println( "MAX AREA: " + findMax( rects, new Comparator( )
            {
                public int compare( Object obj1, Object obj2 )
                {
                    SimpleRectangle r1 = (SimpleRectangle) obj1;
                    SimpleRectangle r2 = (SimpleRectangle) obj2;
                    return( r1.getWidth()*r1.getLength() -  r2.getWidth()*r2.getLength() );
                }
            }
        ) );
    }
}
```

Packages

- Group of related classes.
- Specified by package statement.
- Fewer restrictions on access among each other;
  - if class is called public, then it is visible to all classes
  - if no visibility modifier is specified, it is equivalent to the friend specification from C++, and its visibility is termed as "package visibility" and is somewhere between:
    - private (other classes in package cannot access it) and
    - public (other classes outside package can also access it)
  - A class cannot be private or protected. Only methods & fields are allowed to be declared as such.
- Package locations can be specified by the CLASSPATH environmental variables.
- The import statement helps to get multiple packages. It saves typing.

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.
- A class (except inner classes) cannot be private/protected. But one could make all constructors of a class private.

Access Restrictions of Methods/Fields

- Clients have access to only public methods.
- Derived classes have access to public & protected members of the base class.
- Classes within the same package have access to protected and package members of the base class.

- Public – can be used by anyone .
- Package – by methods of the class and in same package.
- Protected – by methods of the class and subclasses and in the same package.
- Private – only by members of the same class.

Algorithm Analysis

```
public final class MaxSumTest
{  // Fig 5.4, p155
   static private int seqStart = 0;
   static private int seqEnd = -1;
   public static int maxSubSum1( int [ ] a )
   {
      int maxSum = 0;

      for( int i = 0; i < a.length; i++ )
         for( int j = i; j < a.length; j++ )
         {
            int thisSum = 0;

            for( int k = i; k <= j; k++ )
               thisSum += a[ k ];

            if( thisSum > maxSum )
            {
               maxSum   = thisSum;
               seqStart = i;
               seqEnd   = j;
            }
         }

      return maxSum;
   }
}
```

```
public final class MaxSumTest
{  // Fig 5.5, p157
   public static int maxSubSum2( int [ ] a )
   {
      int maxSum = 0;

      for( int i = 0; i < a.length; i++ )
      {
         int thisSum = 0;
         for( int j = i; j < a.length; j++ )
         {
            thisSum += a[ j ];

            if( thisSum > maxSum )
            {
               maxSum = thisSum;
               seqStart = i;
               seqEnd   = j;
            }
         }
      }

      return maxSum;
   }
}
```

Algorithm Analysis

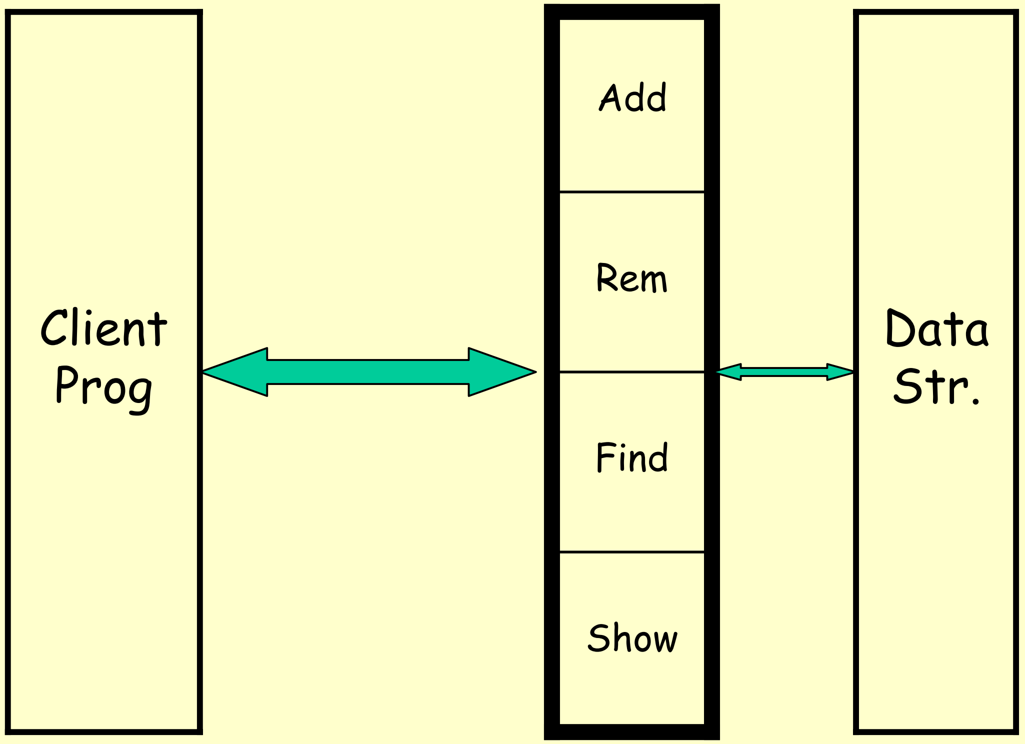
```
public final class MaxSumTest
{  // Fig 5.8, p160
   public static int maxSubSum3( int [ ] a )
   {
      int maxSum = 0;
      int thisSum = 0;

      for( int i = 0, j = 0; j < a.length; j++ )
      {
         thisSum += a[ j ];

         if( thisSum > maxSum )
         {
            maxSum = thisSum;
            seqStart = i;
            seqEnd   = j;
         }
         else if( thisSum < 0 )
         {
            i = j + 1;
            thisSum = 0;
         }
      }

      return maxSum;
   }
}
```

Algorithm Analysis

```
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



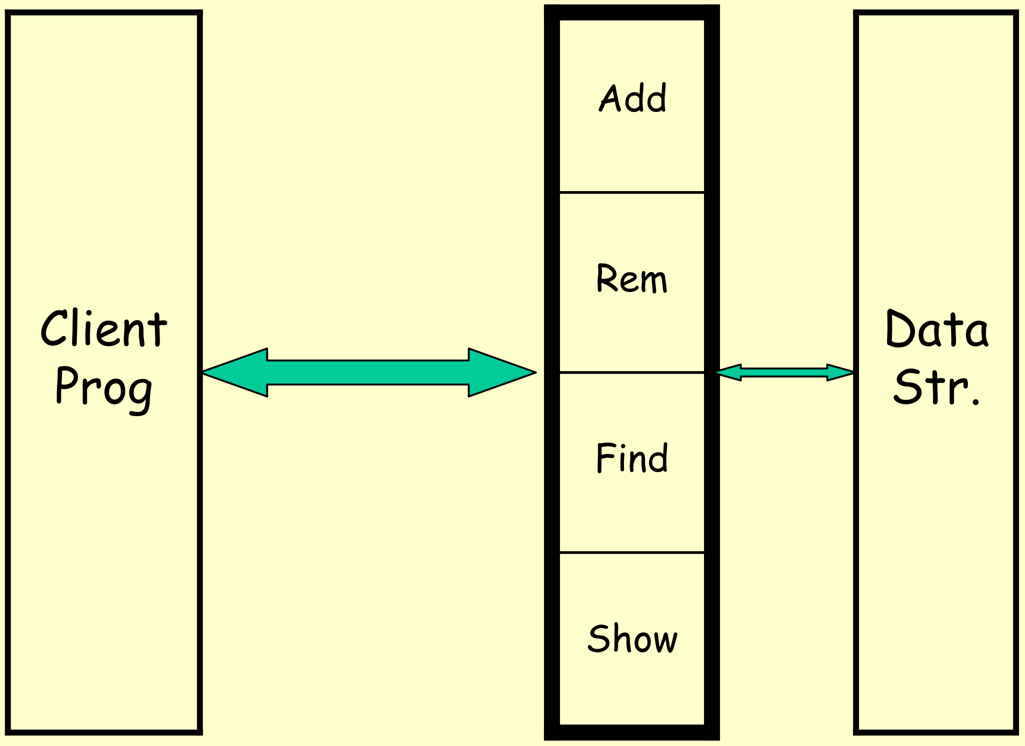
Containers

- Powerful tool for programming data structures
- Provides a library of container classes to “hold your objects”
- 2 types of Containers:
  - Collection: to hold a group of elements e.g., List, Set
  - Map: a group of key-value object pairs. It helps to return “Set of keys, collection of values, set of pairs. Also works with multiple dimensions (i.e., map of maps).
- Iterators give you a better handle on containers and helps to iterate through all the elements. It can be used without any knowledge of how the collection is implemented.
- Collections API provides a few general purpose algorithms that operate on all containers.

```
// Fig 6.9, 6.10, pg 192, 194.
package weiss.util;

public interface Collection extends java.io.Serializable
{
   int size( );
   boolean isEmpty( );
   boolean contains( Object x );
   boolean add( Object x );
   boolean remove( Object x );
   void clear( );
   Iterator iterator( );
   Object [ ] toArray( );
}

public interface Iterator
{
   boolean hasNext( );
   Object next( );
   void remove( );
}
```

```
// Fig 6.11, pg 195
public static void printCollection
              (Collection c)
{
    Iterator itr = c.iterator();
    while (itr.hasNext())
        System.out.println(itr.next());
}
```

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( );
}
```