Access Restrictions of Methods/Fields

Derived Class

Base Class

M1
M2
M3

Package Class

M4

Client Program
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.

- **Private** - only to methods in the same class.
- **Public** - to any method or client program.
- **Package** - to methods in the same package.
- **Protected** - to methods in the same class, in derived classes, and in the same package.
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**.
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( ) ) );
}
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( ) ) );

  // 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];
    }
  }
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.
public final class MaxSumTest
{
    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
{
    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

O(n^3)
Improved Algorithm: 2 Observations

Fix the start (s) and end (e). Assume it results in a positive sum for every block ending at e or to the left of e.

No other start (s’) with the same end (e) can result in a larger sum.

Fix the start (s) and end (e). Assume it results in a positive sum for every block ending to the left of e and a negative sum for the block ending at e.

No other start (s’) between s and e can result in a larger sum.
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;
   }
}
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 ) ) );
    }
}