Wrappers and Adapters

- A wrapper class stores an entity and adds operations that the original type did not support correctly. Java has wrapper types for the 8 primitive types (e.g., Integer for int)
- An adapter class is used when the interface of an existing class needs to be changed to a more appropriate one (e.g., InputStreamReader and OutputStreamWriter that convert byte-oriented streams to character-oriented streams).

```
public class SimpleArrayList  // Fig 4.23, Pg 121, Example of a Wrapper
{
   public SimpleArrayList( )
   {  clear( ); }

    public int size( )
   {  return theSize;  }

   public Object get( int idx )
   {  if( idx < 0 || idx >= size( ) )
         throw new ArrayIndexOutOfBoundsException( "Index " + idx + "; size " + size( ) );
      return theItems[ idx ];
   }

   public boolean add( Object x )
   {  if( theItems.length == size( ) )
      {
         Object [ ] old = theItems;
         theItems = new Object[ theItems.length * 2 + 1 ];
         for( int i = 0; i < size( ); i++ )
            theItems[ i ] = old[ i ];
      }
      theItems[ theSize++ ] = x;
      return true;
   }
   private int theSize;
   private Object [ ] theItems;
}
```

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, 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)
- Package locations can be specified by the CLASSPATH environmental variables.
- The import statement helps to get multiple packages. It saves typing.

Exceptions

- An exception is an object that is thrown from the site of an error and can be caught by an appropriate exception handler.
- Separating the handler from error detection makes the code easier to read and write. finally clause helps cleanup.
- User-defined exceptions can be created or thrown. They are normally not caught in the same block, but passed up to a calling block. For e.g.,

  ```
  throw new NullPointerException();
  ```

- The try region is a guarded region from which errors can be caught by exceptions. Code that good generate an exception is enclosed in a try region. Method is exited if exceptions are thrown from outside try regions. Thus, there is more reliable error recovery without simply exiting.
- It is also possible to rethrow exceptions.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class DivideByTwo
{
   public static void main( String [ ] args )
   {
      BufferedReader in = new BufferedReader( new
                        InputStreamReader( System.in ) );
      int x;
      String oneLine;

      System.out.println( "Enter an integer: " );
      try
      {
         oneLine = in.readLine( );
         x = Integer.parseInt( oneLine );
         System.out.println( "Half of x is " + ( x / 2 ) );
      }
      catch( IOException e )
        { System.out.println( e ); }
      catch( NumberFormatException e )
        { System.out.println( e ); }
   }
}
```

RuntimeExceptions

- Automatically thrown (no need to explicitly throw them).
- No need to specify explicitly that a method might throw one of these exceptions.
- No need to catch them, dealt with automatically.
- It is possible to explicitly throw a runtime exception.

Javadoc

- In C++ specifications are put in .h files and implementations in .cpp files. In Java, only interfaces are put in separate files.
- Appropriate documentation is added to the implementation, and then we run javadoc program to automatically generate a set of HTML files as documentation for the code.
- Javadoc comments start are delimited by /** and **/. Other useful comments are prefaced by @param, @author, @return, @throws.

Algorithm Running Times

Function	Name	Big-Oh
c	Constant	$O(1)$
log N	Logarithmic	$O(\log N)$
$\log^2 N$, $\log^k N$	Log-squared, Poly-logarithmic	$O(\log^2 N)$, $O(\log^k N)$
N	Linear	$O(N)$
N^2 , N^2	Quadratic, Cubic	$O(N^2)$, $O(N^3)$
N^k	Polynomial	$O(N^k)$
2^N	Exponential	$O(2^N)$
2^{2^N}	Super-exponential	$O(2^{2^N})$

```
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;
   }
}
```

```
public final class MaxSumTest
{
   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
{
   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 ) ) );

   }
}
```