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.

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.

```
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;
   }
}
```

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());
}
```