UNIVERSIDAD NACIONAL DE COLOMBIA

Estructuras de Datos

Sesión 2

List Data Structure (Part 2)

Yoan Pinzón

Table of Content Session 2

Linear List Data Structure

Extended version of ArrayLinearList

We can extend the functionality of the original ArrayLinearList beyond its ADT definition. Here we show how to provide the following new methods:

- **Save** : Allows to save the list element into a given file.
- **Load** : Allows to load the list element from a given file.
- **Sort** : Allows to sort the list element by default using `Comparable` or by different keys using `Comparator`.

The new class will be named `ArrayLinearListImproved`

Interface Definition of LinearListImproved

```
package unal.datastructures;

import java.util.*;
import java.io.*;

public interface LinearListImproved<T> extends LinearList<T>
{
   void save( String fn );
   void load( String fn );
   void sort( );
   void sort( Comparator<T> c );
}
```

Class Definition of ArrayLinearListImproved

```
package unal.datastructures;

import java.util.*;
import java.io.*;

public class ArrayLinearListImproved<T extends Serializable & ↙
  ↪ Comparable<? super T>> extends ArrayLinearList<T> implements ↙
  ↪ LinearListImproved<T>
{
   // constructors
   public ArrayLinearListImproved(int initialCapacity) { /* ... */ }
   public ArrayLinearListImproved( ) { /* ... */ }

   // methods
   public void save( String fn ) { /* ... */ }
   public void load( String fn ) { /* ... */ }
```

```
   public void sort( ) { /* ... */ }
   public void sort( Comparator<T> c ) { /* ... */ }
   public static void main( String[] args ) { /* ... */ }
}
```

constructors

```java
public ArrayLinearListImproved(int initialCapacity)
{
   super( initialCapacity );
}

public ArrayLinearListImproved( )
{
   this( 10 );
}
```

A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass

save

```java
/** Save the list into a file */
public void save( String fn )
{
   try( ObjectOutputStream os = new
      ObjectOutputStream( new FileOutputStream( fn ) ) )
   {
      os.writeInt( size );
      for( T x : this ) os.writeObject( x );
   }
   catch( IOException e )
   {
      e.printStackTrace( );
   }
   System.out.println( "Save␣done" );
}
```

load

```java
/** Load from a file into the list.
 * The list is not reset beforehand */
@SuppressWarnings( "unchecked" )
public void load ( String fn )
{
   try( ObjectInputStream is = new
      ObjectInputStream( new FileInputStream( fn ) ) )
   {
      int n = is.readInt( );
      for( int i = 0; i < n; i++ )
         add( i, ( T ) is.readObject( ) );
   }
   catch( IOException | ClassNotFoundException e )
   {
      e.printStackTrace( );
   }
   System.out.println( "Load␣done" );
}
```

sort

```java
/** sort the list using default compareTo */
public void sort ( )
{
   Arrays.sort( element, 0, size );
}

/** sort the list using specific comparator */
public void sort ( Comparator<T> c )
{
   Arrays.sort( element, 0, size, c );
}
```

Comparable implementations provide a natural ordering for a class, which allows objects of that class to be sorted automatically

Java *Comparators* can be passed to a sort method (such as Collections.sort or Arrays.sort) to allow precise control over the sort order

main

```java
/** test program */
public static void main( String[] args )
{
   Random r = new Random( new Date( ).getTime( ) );

   ArrayLinearListImproved<Student> x =
      new ArrayLinearListImproved<>( );
   ArrayLinearListImproved<Student> y =
      new ArrayLinearListImproved<>( );

   x.add( 0, new Student( r.nextInt( 999 ), "Ingrid" ) );
   x.add( 1, new Student( 333, "Zenon" ) );
   x.add( 2, new Student( r.nextInt( 999 ), "Mary" ) );
   x.add( 3, new Student( r.nextInt( 999 ), "Aiden" ) );

   System.out.println( "list␣is␣" + x );
   x.sort( );
   System.out.println( "by␣default␣" + x );
   x.sort( Student.BY_NAME );
```

```java
   System.out.println( "by␣name␣" + x );
   x.sort( Student.BY_NAME_REV );
   System.out.println( "by␣name␣reverse␣" + x );
   x.sort( Student.BY_CODE );
   System.out.println( "by␣code␣" + x );
   x.sort( Student.BY_CODE_REV );
   System.out.println( "by␣code␣reverse␣" + x );
   x.save( "x.dat" );
   y.load( "x.dat" );
   for( Student s : y ) System.out.println( s );
   System.out.println( y.indexOf( new Student( 333 ) ) );
}
```

Compiling `ArrayLinearListImproved.java`

```
C:\2016699\code> javac unal\datastructures\ArrayLinearListImproved.java ↵
C:\2016699\code> java unal.datastructures.ArrayLinearListImproved ↵
list is [[872, Ingrid], [333, Zenon], [639, Mary], [588, Aiden]]
by default [[333, Zenon], [588, Aiden], [639, Mary], [872, Ingrid]]
by name [[588, Aiden], [872, Ingrid], [639, Mary], [333, Zenon]]
by name rev [[333, Zenon], [639, Mary], [872, Ingrid], [588, Aiden]]
by code [[333, Zenon], [588, Aiden], [639, Mary], [872, Ingrid]]
by code rev [[872, Ingrid], [639, Mary], [588, Aiden], [333, Zenon]]
Save done
Load done
[872, Ingrid]
[639, Mary]
[588, Aiden]
[333, Zenon]
3
```

Class Definition of Student

```java
class Student implements Serializable, Comparable<Student>
{
   private int code;
   private String name;

   public static final Comparator<Student> BY_NAME = new ByName( );
   public static final Comparator<Student> BY_NAME_REV = new
      ByNameRev( );
   public static final Comparator<Student> BY_CODE = new Bycode( );
   public static final Comparator<Student> BY_CODE_REV = new
      BycodeRev( );

   public Student( )
   {
      this( 0, "unknown" );
   }

   public Student( int c )
   {
      this( c, "unknown" );
   }
```

```java
public Student( int c, String n )
{
   code = c; name = n;
}

public int getCode( )
{
   return code;
}

public String getName( )
{
   return name;
}

@Override
public String toString( )
{
   return "[" + code + ",␣" + name + "]";
}
```

```java

@Override
public boolean equals( Object o )
{
   if( o == null ) return false;
   if( o == this ) return true;
   if( ! ( o instanceof Student ) ) return false;
   return this.code == ( ( Student ) o ).code;
}

@Override
public int hashCode( )
{
   return Object.hash( code );
}

@Override
public int compareTo( Student o )
{
   return this.code - o.code;
}
```

```java
  private static class ByName implements Comparator<Student>{
    public int compare( Student a, Student b ) {
      return a.getName( ).compareTo( b.getName( ) );
    }
  }

  private static class ByNameRev implements Comparator<Student>{
    public int compare( Student a, Student b ) {
      return -1 * a.getName( ).compareTo( b.getName( ) );
    }
  }

  private static class Bycode implements Comparator<Student>{
    public int compare( Student a, Student b ) {
      return a.code - b.code;
    }
  }

  private static class BycodeRev implements Comparator<Student>{
    public int compare( Student a, Student b ) {
```

```java
      return -1 * ( a.code - b.code );
    }
  }
}
```