PRÁCTICA 7. Estructuras de datos: Pilas, Colas y Listas. Especificación con Interfaces. Implementaciones. Aplicaciones.

¿Qué son las estructuras de datos?

Una estructura de datos es una representación de datos junto con las operaciones permitidas sobre dichos datos. Típicamente todas las estructuras de datos permiten inserciones arbitrarias. Las estructuras de datos varían en como permiten el acceso a los miembros del grupo. Algunas permiten tanto accesos como operaciones de borrado arbitrarios. Otras imponen restricciones, tales como permitir el acceso sólo al elemento más recientemente insertado, o al menos recientemente insertado.

Las estructuras de datos nos permiten lograr un importante objetivo de la programación orientada a objetos: la reutilización de componentes. Una vez que una estructura de datos ha sido implementada, puede ser utilizada una y otra vez en diversas aplicaciones.

El enfoque, separación de la interfaz y la implementación, es parte del paradigma de la programación orientada a objetos. El usuario de la estructura de datos no necesita ver la implementación, sólo las operaciones disponibles. Esta es la parte de ocultamiento y encapsulación, de la programación orientada a objetos. Otra parte importante de la programación orientada a objetos es la abstracción, Se deben pensar cuidadosamente el diseño de las estructuras de datos sin tener en cuenta la implementación. Esto hace la interfaz más limpia, más flexible, más reutilizable y generalmente más fácil de implementar.

Nota:

Junto con esta práctica van los siguientes archivos:

1. Las interfaces Pila.java, Cola.java y la clase NodoLista.java. Deberá añadirlas al paquete estructurasdatos.
2. La excepción DesbordamientoInferiorException.java. Deberá añadirla al paquete excepciones
3. Las interfaces Iterable.java, Iterador.java, Lista.java que deberá añadirlas al paquete com.mp.estructurasdatos, también la clase ListaAbstracta. FueraLimitesException.java debe añadir al paquete com.mp.excepciones una vez acabado el ejercicio, junto con las clases de implementación.

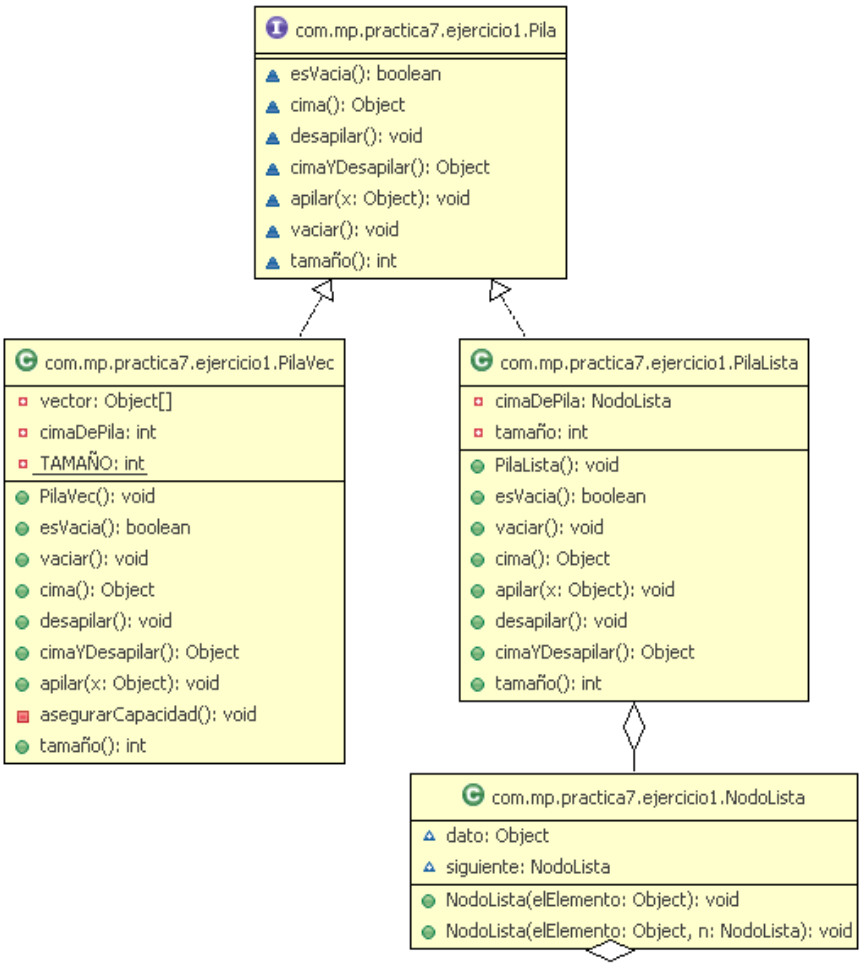
Los distintos juegos de prueba.

1. Consulte y estudie mediante la documentación de la API las siguientes clases del paquete java.util Stack, StringTokenizer
2. Consulte y estudie mediante la documentación de la API del paquete java.util, las siguientes interfaces List, ListIetrator y Collection y las clases ArrayList, LinkedList y Collections.

Conforme vaya realizando los distintos ejercicios, una vez comprobadas las distintas estructuras de datos, debe realizar una copia de las clases e interfaces correspondientes en el paquete **com.mp.estructurasdatos.**

Ejercicios a realizar

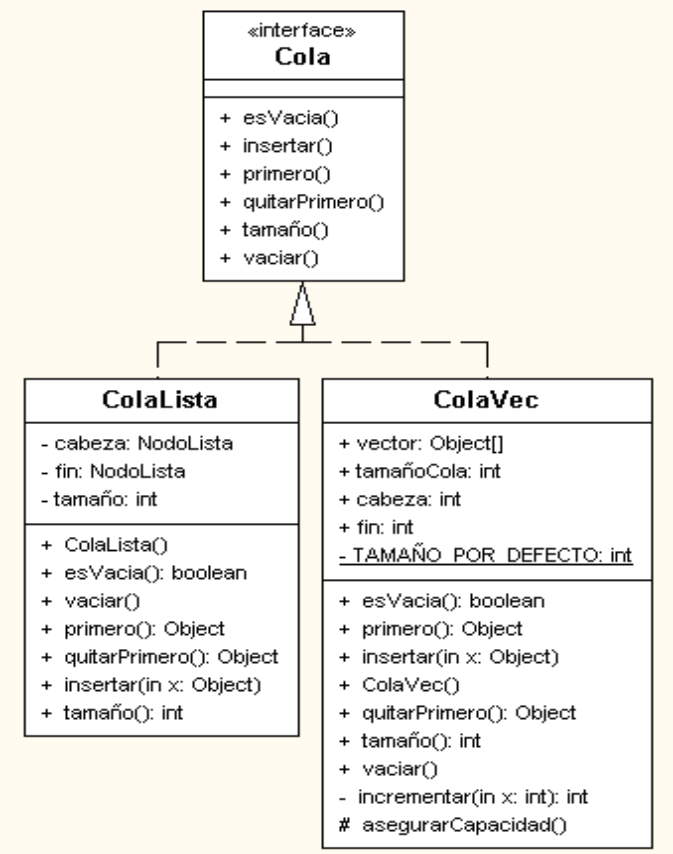
1) Utilizando la especificación de una Pila representada por la interfaz java de pila (Pila.java).
   a) Implemente utilizando vectores la pila (PilaVec.java). Utilice el método privado *asegurarCapacidad()* para incrementar el tamaño de la Pila.
   b) Implemente utilizando listas la pila (PilaLista.java). Utilice la clase NodoLista.



Nota: Incluir en el paquete: **com.mp.practica7.ejercicio1**.

Nota: Los tests se situaran en: **com.mp.practica7.ejercicio1.test.**

2) Utilizando la especificación de una Cola representada por la interfaz java de pila (Cola.java).
   a) Implemente utilizando vectores la cola (ColaVec.java). Construya una implementación circular que devuelve la cabeza y fin al principio del vector cuando sobrepasan la última posición.
   b) Implemente utilizando listas la cola (ColaLista.java). Utilice la clase NodoLista.



Nota: Incluir en el paquete: **com.mp.practica7.ejercicio2**

Nota: Los tests se situarán en: **com.mp.practica7.ejercicio2.test**

3) Diseñe e implemente una calculadora que permita evaluar expresiones aritméticas, con matemática entera. La clase se denominará Evaluador

Los requisitos son:

- Debe evaluar expresiones postfijas mediante una máquina postfija. La entrada será un array de String y la salida un entero con el valor del cálculo. Utilice una pila para generar la expresión postfija. (método postfija).
- Debe convertir expresiones infijas en postfijas realizando la conversión mediante un algoritmo de análisis sintáctico con precedencia de operadores. Esta conversión soportará sumas, restas, multiplicaciones, divisiones. También evaluará los paréntesis. Utilice una pila para convertir la expresión infija en postfija. (método infijaPostfija).
- Un método para realizar la división en tokens, al que se le pasa una cadena con una expresión aritmética y devuelva un array de String con los símbolos. Utilice la clase StringTokenizer del paquete java.util para realizar la división en tokens que indican los símbolos considerados como delimitadores. (método parser).

Nota: Incluir en el paquete: **com.mp.practica7.ejercicio3**

Nota: Los tests se situaran en: **com.mp.practica7.ejercicio3.test**

4) Dada la siguiente especificación textual: "Se desea modelar una parada de taxis. Sobre la parada se desea saber el nombre de la parada y la dirección donde está situada. Sobre los taxis se desea guardar en una primera versión sólo su número de licencia y si es adatado para minusválidos. Por defecto la parada de taxis esta vacía, y a esta van llegando taxis. Van saliendo taxis, a petición de los clientes. El orden de salida es el de una cola, el primero en llegar es el primero en salir. Si un cliente solicita un taxi adaptado para minusválidos saldrá el primer taxi que cumpla esta condición saltándose el orden establecido. También seria útil saber el número de taxis en la parada y un listado en pantalla de los taxis que están en la parada". Diseñe e implemente una aplicación en Java que resuelva la especificación. Utilice una estructura de datos Cola.

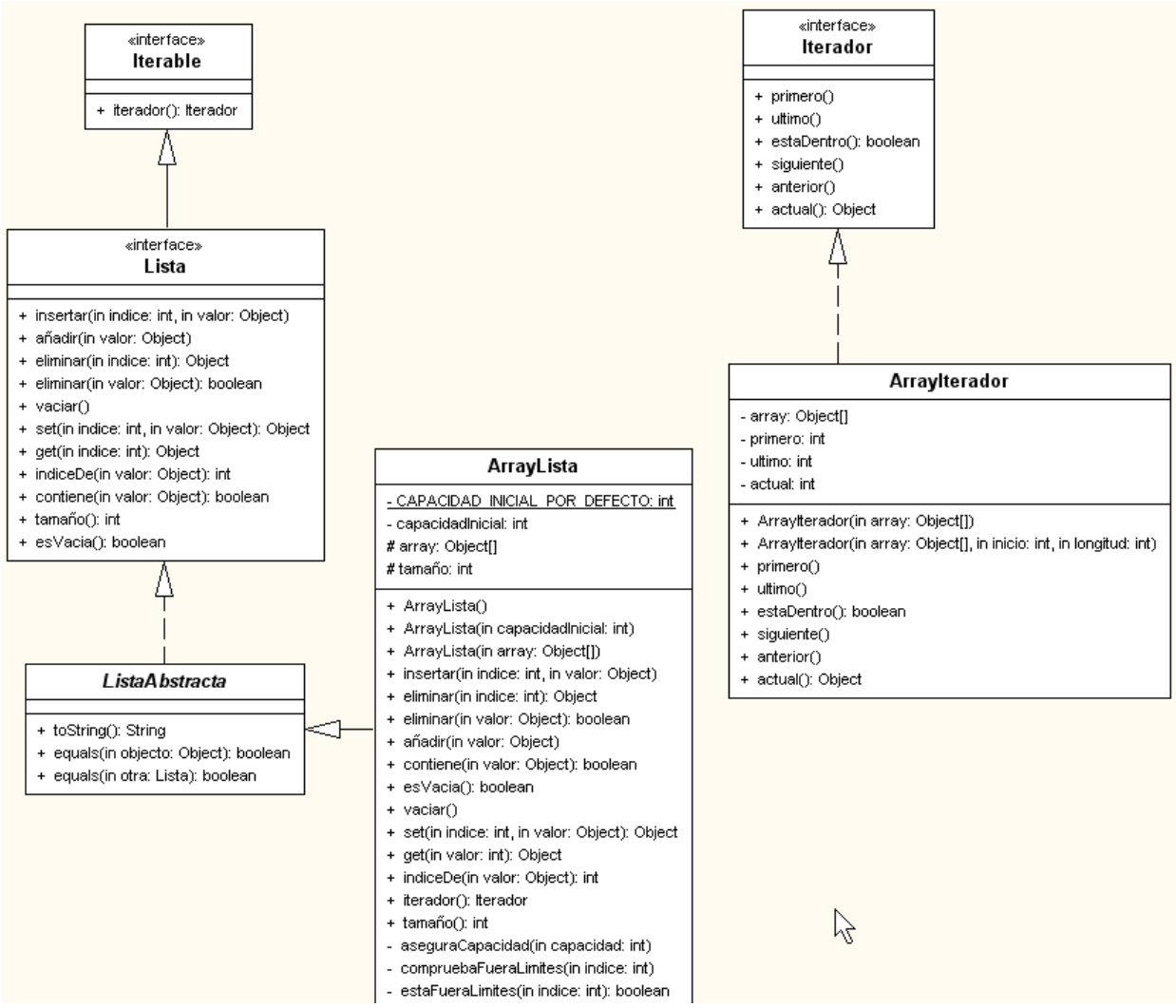
   Nota: Incluir en el paquete: **com.mp.practica7.ejercicio4**

   Nota: Los tests se situaran en: **com.mp.practica7.ejercicio4.test**

5) Implementación de una lista utilizando un array. Utilice las interfaces Iterable, Iterador y Lista y la clase abstracta ListaAbstracta (que acompañan a la práctica). Las clases que el alumno tiene que implementar son ArrayIterator y ArrayLista**.**

   Nota: Incluir en el paquete: com.mp.practica7.ejercicio5

   Nota: Los tests se situaran en: **com.mp.practica7.ejercicio5.test**

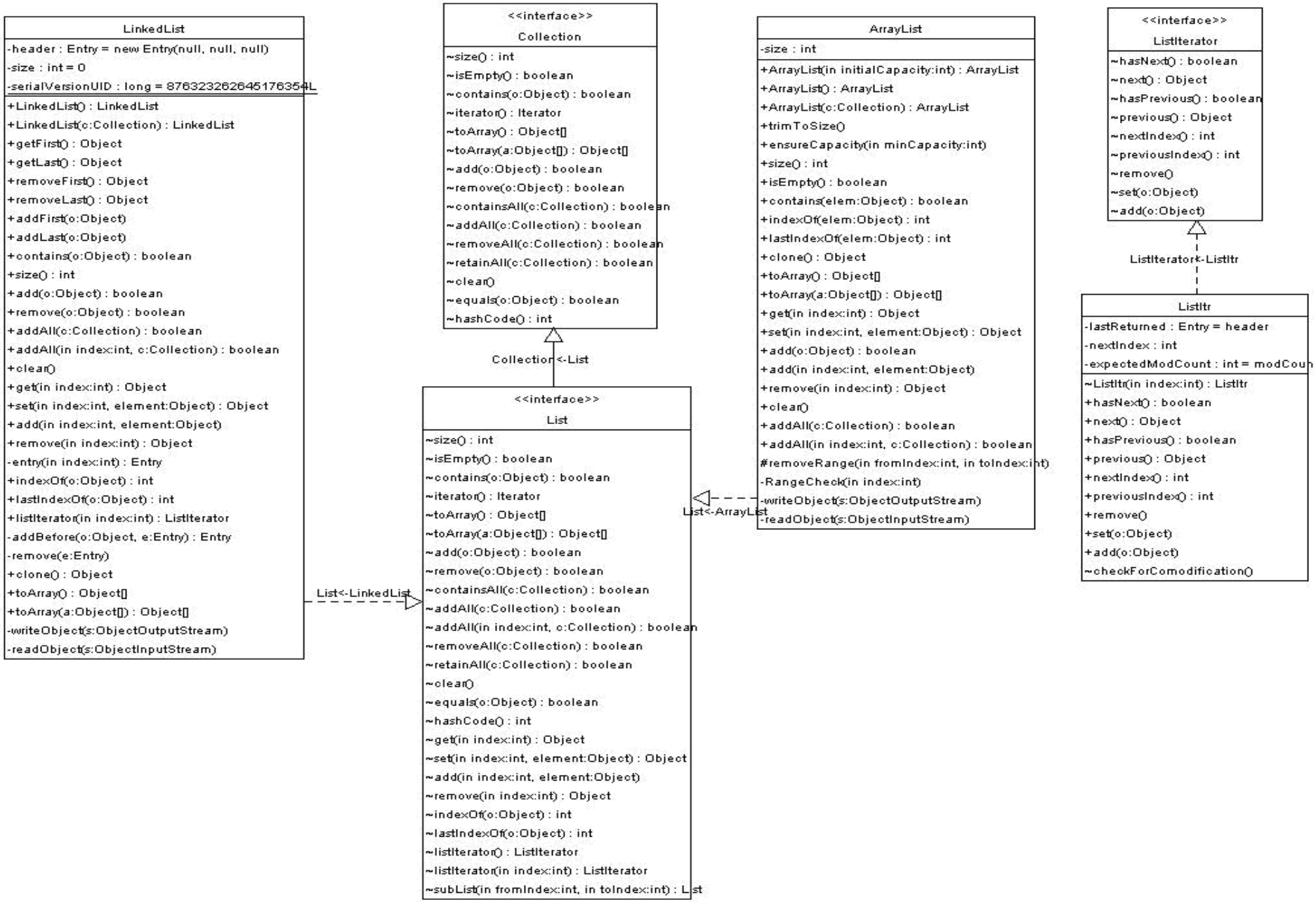


6) Revisión del ejercicio quinto de la segunda práctica (“residencia”).Manteniendo la misma funcionalidad, debe cambiar la implementación utilizando listas para representar las repeticiones mantenidas por arrays. Deberá utilizar la clase ArrayList de java.util para las propiedades cuidadores y habitaciones de la clase Residencia y las clase ArrayLista (implementada en el ejercicio anterior) para la propiedad residentes y finalmente LinkedList del paquete java.util. para la propiedad reservas.

   Nota: Incluir en el paquete: **com.mp.practica7.ejercicio6**

   Nota: Los tests se situaran en: **com.mp.practica7.ejercicio6.test**

7) Genere el diagrama de clases para el paquete com.mp.estructurasdatos utilizando el plugin AmaterasUML.


LinkedList
-header : Entry = new Entry(null, null, null)
-size : int = 0
-serialVersionUID : long = 876323262645176354L
+LinkedList() : LinkedList
+LinkedList(c:Collection) : LinkedList
+getFirst() : Object
+getLast() : Object
+removeFirst() : Object
+removeLast() : Object
+addFirst(o:Object)
+addLast(o:Object)
+contains(o:Object) : boolean
+size() : int
+add(o:Object) : boolean
+remove(o:Object) : boolean
+addAll(c:Collection) : boolean
+addAll(in index:int, c:Collection) : boolean
+clear()
+get(in index:int) : Object
+set(in index:int, element:Object) : Object
+add(in index:int, element:Object)
+remove(in index:int) : Object
-entry(in index:int) : Entry
+indexOf(o:Object) : int
+lastIndexOf(o:Object) : int
+listIterator(in index:int) : ListIterator
-addBefore(o:Object, e:Entry) : Entry
-remove(e:Entry)
+clone() : Object
+toArray() : Object[]
+toArray(a:Object[]) : Object[]
-writeObject(s:ObjectOutputStream)
-readObject(s:ObjectInputStream)
List<-LinkedList
<<interface>>
Collection
~size() : int
~isEmpty() : boolean
~contains(o:Object) : boolean
~iterator() : Iterator
~toArray() : Object[]
~toArray(a:Object[]) : Object[]
~add(o:Object) : boolean
~remove(o:Object) : boolean
~containsAll(c:Collection) : boolean
~addAll(c:Collection) : boolean
~removeAll(c:Collection) : boolean
~retainAll(c:Collection) : boolean
~clear()
~equals(o:Object) : boolean
~hashCode() : int
Collection<-List
<<interface>>
List
~size() : int
~isEmpty() : boolean
~contains(o:Object) : boolean
~iterator() : Iterator
~toArray() : Object[]
~toArray(a:Object[]) : Object[]
~add(o:Object) : boolean
~remove(o:Object) : boolean
~containsAll(c:Collection) : boolean
~addAll(c:Collection) : boolean
~addAll(in index:int, c:Collection) : boolean
~removeAll(c:Collection) : boolean
~retainAll(c:Collection) : boolean
~clear()
~equals(o:Object) : boolean
~hashCode() : int
~get(in index:int) : Object
~set(in index:int, element:Object) : Object
~add(in index:int, element:Object)
~remove(in index:int) : Object
~indexOf(o:Object) : int
~lastIndexOf(o:Object) : int
~listIterator() : ListIterator
~listIterator(in index:int) : ListIterator
~subList(in fromIndex:int, in toIndex:int) : List
List<-ArrayList
ArrayList
-size : int
+ArrayList(in initialCapacity:int) : ArrayList
+ArrayList() : ArrayList
+ArrayList(c:Collection) : ArrayList
+trimToSize()
+ensureCapacity(in minCapacity:int)
+size() : int
+isEmpty() : boolean
+contains(elem:Object) : boolean
+indexOf(elem:Object) : int
+lastIndexOf(elem:Object) : int
+clone() : Object
+toArray() : Object[]
+toArray(a:Object[]) : Object[]
+get(in index:int) : Object
+set(in index:int, element:Object) : Object
+add(o:Object) : boolean
+add(in index:int, element:Object)
+remove(in index:int) : Object
+clear()
+addAll(c:Collection) : boolean
+addAll(in index:int, c:Collection) : boolean
#removeRange(in fromIndex:int, in toIndex:int)
-RangeCheck(in index:int)
-writeObject(s:ObjectOutputStream)
-readObject(s:ObjectInputStream)
<<interface>>
ListIterator
~hasNext() : boolean
~next() : Object
~hasPrevious() : boolean
~previous() : Object
~nextIndex() : int
~previousIndex() : int
~remove()
~set(o:Object)
~add(o:Object)
ListIterator<-ListItr
ListItr
-lastReturned : Entry = header
-nextIndex : int
-expectedModCount : int = modCoun
~ListItr(in index:int) : ListItr
+hasNext() : boolean
+next() : Object
+hasPrevious() : boolean
+previous() : Object
+nextIndex() : int
+previousIndex() : int
+remove()
+set(o:Object)
+add(o:Object)
~checkForComodification()