INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERÍA

CAMPUS ZACATECAS

PROGRAMACIÓN ORIENTADA A OBJETOS

LISTAS EN JAVA

MAURO TALAMANTES VILLAGRANA

ROBERTO OSWALDO CRUZ LEIJA

24/10/19

List

Extiende de collection.

Implementa AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector.

Una colección ordenada (también conocida como secuencia). El usuario de esta interfaz tiene un control preciso sobre dónde se inserta cada elemento en la lista. El usuario puede acceder a los elementos por su índice entero (posición en la lista) y buscar elementos en la lista.

A diferencia de los conjuntos, las listas generalmente permiten elementos duplicados. Más formalmente, las listas generalmente permiten pares de elementos e1 y e2 de modo que e1. equals (e2), y generalmente permiten múltiples elementos nulos si permiten elementos nulos. No es inconcebible que alguien desee implementar una lista que prohíba los duplicados, lanzando excepciones de tiempo de ejecución cuando el usuario intenta insertarlos, pero esperamos que este uso sea raro.

La interfaz Lista coloca estipulaciones adicionales, más allá de las especificadas en la interfaz Colección, en los contratos del iterador, agregar, eliminar, iguales y métodos hashCode. Las declaraciones para otros métodos heredados también se incluyen aquí por conveniencia.

La interfaz Lista proporciona cuatro métodos para el acceso posicional (indexado) a los elementos de la lista. Las listas (como las matrices Java) están basadas en cero. Tenga en cuenta que estas operaciones pueden ejecutarse en un tiempo proporcional al valor del índice para algunas implementaciones (la clase LinkedList, por ejemplo). Por lo tanto, iterar sobre los elementos en una lista generalmente es preferible a indexarlo si la persona que llama no conoce la implementación.

La interfaz List proporciona un iterador especial, llamado ListIterator, que permite la inserción y sustitución de elementos, y el acceso bidireccional, además de las operaciones normales que proporciona la interfaz Iterator. Se proporciona un método para obtener un iterador de lista que comienza en una posición especificada en la lista.

La interfaz de Lista proporciona dos métodos para buscar un objeto específico. Desde el punto de vista del rendimiento, estos métodos deben usarse con precaución. En muchas implementaciones, realizarán búsquedas lineales costosas.

La interfaz de Lista proporciona dos métodos para insertar y eliminar de manera eficiente múltiples elementos en un punto arbitrario de la lista.

Nota: Si bien es permisible que las listas se contengan a sí mismas como elementos, se recomienda precaución extrema: los métodos igual y hashCode ya no están bien definidos en dicha lista.

Algunas implementaciones de listas tienen restricciones sobre los elementos que pueden contener. Por ejemplo, algunas implementaciones prohíben elementos nulos, y algunas tienen restricciones sobre los tipos de sus elementos. Intentar agregar un elemento no elegible genera una excepción no verificada, generalmente NullPointerException o ClassCastException. Intentar consultar la presencia de un elemento no elegible puede arrojar una excepción, o simplemente puede devolver falso; algunas implementaciones exhibirán el comportamiento anterior y algunas exhibirán el último. En términos más generales, intentar una operación en un elemento no elegible cuya finalización no resultaría en la inserción de un elemento no elegible en la lista puede arrojar una excepción o puede tener éxito, a

opción de la implementación. Dichas excepciones están marcadas como "opcionales" en la especificación de esta interfaz.

Esta interfaz es miembro de Java Collections Framework.

Métodos

Modificador y Tipo	Método y descripción
boolean	**add**(**E** e) Agrega el elemento especificado al final de esta lista (operación opcional).
void	**add**(int index, **E** element) Inserta el elemento especificado en la posición especificada en esta lista (operación opcional).
boolean	**addAll**(**Collection**<? extends **E**> c) Agrega todos los elementos de la colección especificada al final de esta lista, en el orden en que los devuelve el iterador de la colección especificada (operación opcional).
boolean	**addAll**(int index, **Collection**<? extends **E**> c) Inserta todos los elementos de la colección especificada en esta lista en la posición especificada (operación opcional).
void	**clear**() Elimina todos los elementos de esta lista (operación opcional).
boolean	**contains**(**Object** o) Devuelve verdadero si esta lista contiene el elemento especificado.
boolean	**containsAll**(**Collection**<?> c) Devuelve verdadero si esta lista contiene todos los elementos de la colección especificada.
boolean	**equals**(**Object** o) Compara el objeto especificado con esta lista para la igualdad.
E	**get**(int index) Devuelve el elemento en la posición especificada en esta lista.
int	**hashCode**() Devuelve el valor del código hash para esta lista.
int	**indexOf**(**Object** o) Devuelve el índice de la primera aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento.
boolean	**isEmpty**() Devuelve verdadero si esta lista no contiene elementos.
Iterator<**E**>	**iterator**() Devuelve un iterador sobre los elementos de esta lista en la secuencia adecuada.
int	**lastIndexOf**(**Object** o)Devuelve el índice de la última aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento.

ListIterator<E>	listIterator() Devuelve un iterador de lista sobre los elementos de esta lista (en la secuencia adecuada).
ListIterator<E>	listIterator(int index) Devuelve un iterador de lista sobre los elementos de esta lista (en la secuencia adecuada), comenzando en la posición especificada en la lista.
E	remove(int index) Elimina el elemento en la posición especificada en esta lista (operación opcional).
boolean	remove(Object o) Elimina la primera aparición del elemento especificado de esta lista, si está presente (operación opcional).
boolean	removeAll(Collection<?> c) Elimina de esta lista todos sus elementos contenidos en la colección especificada (operación opcional).
boolean	retainAll(Collection<?> c) Retiene solo los elementos de esta lista que están contenidos en la colección especificada (operación opcional).
E	set(int index, E element) Reemplaza el elemento en la posición especificada en esta lista con el elemento especificado (operación opcional).
int	size() Devuelve el número de elementos en esta lista.
List<E>	subList(int fromIndex, int toIndex) Devuelve una vista de la parte de esta lista entre la especificada fromIndex , inclusive, y toIndex , exclusive.
Object[]	toArray() Devuelve una matriz que contiene todos los elementos de esta lista en la secuencia adecuada (del primer al último elemento).
<T> T[]	toArray(T[] a) Devuelve una matriz que contiene todos los elementos de esta lista en la secuencia adecuada (del primer al último elemento); El tipo de tiempo de ejecución de la matriz devuelta es el de la matriz especificada.

- ArrayList.

public class ArrayList <E> extiende AbstractList <E>

implementa List <E>, RandomAccess, Cloneable, Serializable

Implementación de matriz redimensionable de la interfaz List. Implementa todas las operaciones de lista opcionales y permite todos los elementos, incluido nulo. Además de implementar la interfaz de Lista, esta clase proporciona métodos para manipular el tamaño de la matriz que se usa internamente para almacenar la lista. (Esta clase es aproximadamente equivalente a Vector, excepto que no está sincronizada).

El tamaño, estaVacia, obtener, conjunto, iterador, y ListIterator las operaciones se ejecutan en tiempo constante. La operación de agregar se ejecuta en tiempo constante amortizado, es decir, agregar n elementos requiere tiempo O (n). Todas las demás operaciones se ejecutan en tiempo lineal (aproximadamente). El factor constante es bajo en comparación con el de la implementación de LinkedList.

Cada instancia de ArrayList tiene una capacidad. La capacidad es el tamaño de la matriz utilizada para almacenar los elementos en la lista. Siempre es al menos tan grande como el tamaño de la lista. A medida que se agregan elementos a una ArrayList, su capacidad crece automáticamente. Los detalles de la política de crecimiento no se especifican más allá del hecho de que agregar un elemento tiene un costo de tiempo amortizado constante.

Una aplicación puede aumentar la capacidad de una instancia de ArrayList antes de agregar una gran cantidad de elementos mediante la operación de sureCapacity. Esto puede reducir la cantidad de reasignación incremental.

Tenga en cuenta que esta implementación no está sincronizada. Si varios subprocesos acceden a una instancia de ArrayList al mismo tiempo, y al menos uno de los subprocesos modifica la lista estructuralmente, debe sincronizarse externamente. (Una modificación estructural es cualquier operación que agrega o elimina uno o más elementos, o cambia el tamaño explícitamente de la matriz de respaldo; simplemente establecer el valor de un elemento no es una modificación estructural). Esto se logra típicamente mediante la sincronización de algún objeto que encapsula naturalmente lista. Si no existe tal objeto, la lista debe "ajustarse" utilizando el Collections.synchronizedList método Esto se hace mejor en el momento de la creación, para evitar el acceso accidental no sincronizado a la lista:

Lista lista = Colecciones.synchronizedList (nueva ArrayList (...));

Los iteradores devueltos por esta claseiterator y los listIteratormétodos son rápidos: si la lista se modifica estructuralmente en cualquier momento después de que se crea el iterador, de cualquier manera, excepto a través de los métodos removeo iteradores propios add, el iterador arrojará un ConcurrentModificationException. Por lo tanto, frente a la modificación concurrente, el iterador falla de manera rápida y limpia, en lugar de arriesgarse a un comportamiento arbitrario, no determinista en un momento indeterminado en el futuro.

Tenga en cuenta que el comportamiento a prueba de fallas de un iterador no puede garantizarse ya que, en términos generales, es imposible hacer garantías duras en presencia de modificaciones concurrentes no sincronizadas. Los iteradores a prueba de fallas se ConcurrentModificationExceptionbasan en el mejor esfuerzo. Por lo tanto, sería un error escribir un programa que dependiera de esta excepción para su corrección: el comportamiento rápido de los iteradores debe usarse solo para detectar errores.

Esta clase es miembro de Java Collections Framework.

Constructores	
Constructor y Descripción	
ArrayList()	Construye una lista vacía con una capacidad inicial de diez.
ArrayList(**Collection**<? extends **E**> c)	Construye una lista que contiene los elementos de la colección especificada, en el orden en que los devuelve el iterador de la colección.
ArrayList(int initialCapacity)	Construye una lista vacía con la capacidad inicial especificada.

Métodos

Modificador y Tipo	Método y descripción
`boolean`	**add**(**E** e) Agrega el elemento especificado al final de esta lista.
`void`	**add**(int index, **E** element) Inserta el elemento especificado en la posición especificada en esta lista.
`boolean`	**addAll**(**Collection**<? extends **E**> c) Agrega todos los elementos de la colección especificada al final de esta lista, en el orden en que los devuelve el iterador de la colección especificada.
`boolean`	**addAll**(int index, **Collection**<? extends **E**> c) Inserta todos los elementos de la colección especificada en esta lista, comenzando en la posición especificada.
`void`	**clear**() Elimina todos los elementos de esta lista.
Object	**clone**() Devuelve una copia superficial de esta instancia de `ArrayList` .
`boolean`	**contains**(**Object** o) Devuelve `verdadero` si esta lista contiene el elemento especificado.
`void`	**ensureCapacity**(int minCapacity) Aumenta la capacidad de esta instancia de `ArrayList`, si es necesario, para garantizar que pueda contener al menos el número de elementos especificados por el argumento de capacidad mínima.
E	**get**(int index) Devuelve el elemento en la posición especificada en esta lista.
`int`	**indexOf**(**Object** o) Devuelve el índice de la primera aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento.
`boolean`	**isEmpty**() Devuelve `verdadero` si esta lista no contiene elementos.
Iterator<**E**>	**iterator**() Devuelve un iterador sobre los elementos de esta lista en la secuencia adecuada.
`int`	**lastIndexOf**(**Object** o) Devuelve el índice de la última aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento.
ListIterator<**E**>	**listIterator**() Devuelve un iterador de lista sobre los elementos de esta lista (en la secuencia adecuada).
ListIterator<**E**>	**listIterator**(int index) Devuelve un iterador de lista sobre los elementos de esta lista (en la secuencia adecuada), comenzando en la posición especificada en la lista.
E	**remove**(int index) Elimina el elemento en la posición especificada en esta lista.

`boolean`	**remove**(**Object** o) Elimina la primera aparición del elemento especificado de esta lista, si está presente.
`boolean`	**removeAll**(**Collection**<?> c) Elimina de esta lista todos sus elementos contenidos en la colección especificada.
`protected void`	**removeRange**(int fromIndex, int toIndex) Elimina de esta lista todos los elementos cuyo índice está entre `fromIndex`, inclusivo y `toIndex`exclusivo.
`boolean`	**retainAll**(**Collection**<?> c) Retiene solo los elementos de esta lista contenidos en la colección especificada.
E	**set**(int index, **E** element) Reemplaza el elemento en la posición especificada en esta lista con el elemento especificado.
`int`	**size**() Devuelve el número de elementos en esta lista.
List<**E**>	**subList**(int fromIndex, int toIndex) Devuelve una vista de la parte de esta lista entre lo especificado `fromIndex`, inclusivo y `toIndex`exclusivo.
Object[]	**toArray**() Devuelve una matriz que contiene todos los elementos de esta lista en la secuencia adecuada (del primer al último elemento).
<T> T[]	**toArray**(T[] a) Devuelve una matriz que contiene todos los elementos de esta lista en la secuencia adecuada (del primer al último elemento); El tipo de tiempo de ejecución de la matriz devuelta es el de la matriz especificada.
`void`	**trimToSize**() Recorta la capacidad de esta instancia de `ArrayList` para que sea el tamaño actual de la lista.

LinkedList.

clase pública LinkedList <E> extiende AbstractSequentialList <E>

implementa List <E>, Deque <E>, Cloneable, Serializable

Implementación de lista doblemente vinculada de las interfaces Listy Deque. Implementa todas las operaciones de lista opcionales y permite todos los elementos (incluidos null).

Todas las operaciones funcionan como podría esperarse para una lista doblemente vinculada. Las operaciones que se indexan en la lista atravesarán la lista desde el principio o el final, lo que esté más cerca del índice especificado.

Tenga en cuenta que esta implementación no está sincronizada. Si varios subprocesos acceden a una lista vinculada al mismo tiempo, y al menos uno de los subprocesos modifica la lista estructuralmente, debe sincronizarse externamente. (Una modificación estructural es cualquier operación que agrega o elimina uno o más elementos; simplemente establecer el valor de un elemento no es una modificación estructural). Esto generalmente se logra mediante la sincronización de algún objeto que encapsula naturalmente la lista. Si no existe tal objeto, la lista debe "ajustarse" utilizando el Collections.synchronizedList método Esto se hace mejor en el momento de la creación, para evitar el acceso accidental no sincronizado a la lista:

```
Lista lista = Colecciones.synchronizedList (nueva LinkedList (...));
```

Los iteradores devueltos por esta clase iteratory listIteratormétodos son rápidos: si la lista se modifica estructuralmente en cualquier momento después de que se crea el iterador, de cualquier manera, excepto a través del método removeo iteradores propios add, el iterador arrojará un ConcurrentModificationException. Por lo tanto, frente a la modificación concurrente, el iterador falla de manera rápida y limpia, en lugar de arriesgarse a un comportamiento arbitrario, no determinista en un momento indeterminado en el futuro.

Tenga en cuenta que el comportamiento a prueba de fallas de un iterador no puede garantizarse ya que, en términos generales, es imposible hacer garantías duras en presencia de modificaciones concurrentes no sincronizadas. Los iteradores a prueba de fallas se ConcurrentModificationExceptionbasan en el mejor esfuerzo. Por lo tanto, sería un error escribir un programa que dependiera de esta excepción para su corrección: el comportamiento rápido de los iteradores debe usarse solo para detectar errores.

Esta clase es miembro de Java Collections Framework.

Constructores

Constructor y Descripción
LinkedList() Construye una lista vacía.
LinkedList(**Collection**<? extends **E**> c) Construye una lista que contiene los elementos de la colección especificada, en el orden en que los devuelve el iterador de la colección.

Métodos

Modificador y Tipo	Método y descripción
boolean	**add**(**E** e) Agrega el elemento especificado al final de esta lista.
void	**add**(int index, **E** element) Inserta el elemento especificado en la posición especificada en esta lista.
boolean	**addAll**(**Collection**<? extends **E**> c) Agrega todos los elementos de la colección especificada al final de esta lista, en el orden en que los devuelve el iterador de la colección especificada.
boolean	**addAll**(int index, **Collection**<? extends **E**> c) Inserta todos los elementos de la colección especificada en esta lista, comenzando en la posición especificada.
void	**addFirst**(**E** e) Inserta el elemento especificado al principio de esta lista.
void	**addLast**(**E** e) Agrega el elemento especificado al final de esta lista.
void	**clear**() Elimina todos los elementos de esta lista.
Object	**clone**() Devuelve una copia superficial de esto LinkedList.
boolean	**contains**(**Object** o) Devuelve truesi esta lista contiene el elemento especificado.
Iterator<**E**>	**descendingIterator**() Devuelve un iterador sobre los elementos en esta deque en orden secuencial inverso.
E	**element**() Recupera, pero no elimina, el encabezado (primer elemento) de esta lista.
E	**get**(int index) Devuelve el elemento en la posición especificada en esta lista.
E	**getFirst**() Devuelve el primer elemento de esta lista.
E	**getLast**() Devuelve el último elemento en esta lista.
int	**indexOf**(**Object** o) Devuelve el índice de la primera aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento.
int	**lastIndexOf**(**Object** o) Devuelve el índice de la última aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento.
ListIterator<**E**>	**listIterator**(int index) Devuelve un iterador de lista de los elementos en esta lista (en la secuencia adecuada), come
boolean	**offer**(**E** e) Agrega el elemento especificado como la cola (último elemento) de esta lista.
boolean	**offerFirst**(**E** e) Inserta el elemento especificado al principio de esta lista.
boolean	**offerLast**(**E** e) Inserta el elemento especificado al final de esta lista.
E	**peek**()

	Recupera, pero no elimina, el encabezado (primer elemento) de esta lista.
E	peekFirst() Recupera, pero no elimina, el primer elemento de esta lista, o devuelve nullsi esta lista está vacía.
E	peekLast() Recupera, pero no elimina, el último elemento de esta lista, o devuelve nullsi esta lista está vacía.
E	poll() Recupera y elimina el encabezado (primer elemento) de esta lista.
E	pollFirst() Recupera y elimina el primer elemento de esta lista, o regresa nullsi esta lista está vacía.
E	pollLast() Recupera y elimina el último elemento de esta lista, o regresa nullsi esta lista está vacía.
E	pop() Hace estallar un elemento de la pila representada por esta lista.
void	push(E e) Empuja un elemento en la pila representada por esta lista.
E	remove() Recupera y elimina el encabezado (primer elemento) de esta lista.
E	remove(int index) Elimina el elemento en la posición especificada en esta lista.
boolean	remove(Object o) Elimina la primera aparición del elemento especificado de esta lista, si está presente.
E	removeFirst() Elimina y devuelve el primer elemento de esta lista.
boolean	removeFirstOccurrence(Object o) Elimina la primera aparición del elemento especificado en esta lista (al recorrer la lista de principio a fin).
E	removeLast() Elimina y devuelve el último elemento de esta lista.
boolean	removeLastOccurrence(Object o) Elimina la última aparición del elemento especificado en esta lista (al recorrer la lista de principio a fin).
E	set(int index, E element) Reemplaza el elemento en la posición especificada en esta lista con el elemento especificado.
int	size() Devuelve el número de elementos en esta lista.
Object[]	toArray() Devuelve una matriz que contiene todos los elementos de esta lista en la secuencia adecuada (del primer al último elemento).
<T> T[]	toArray(T[] a) Devuelve una matriz que contiene todos los elementos de esta lista en la secuencia adecuada (del primer al último elemento); El tipo de tiempo de ejecución de la matriz devuelta es el de la matriz especificada.