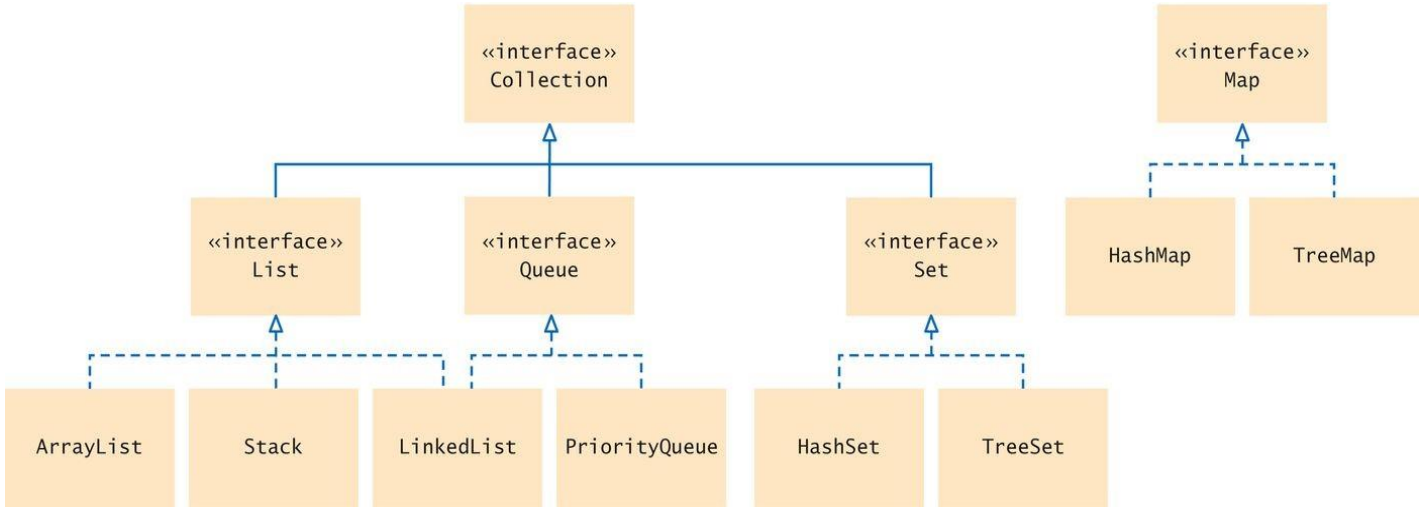
Java kolekcijos

LINKED LIST

Java kolekcijos

JAVA turi įvairių kolekcijų skirtų darbui su duomenimis. Jos visos turi savo paskirtį, privalumus ir trūkumus.

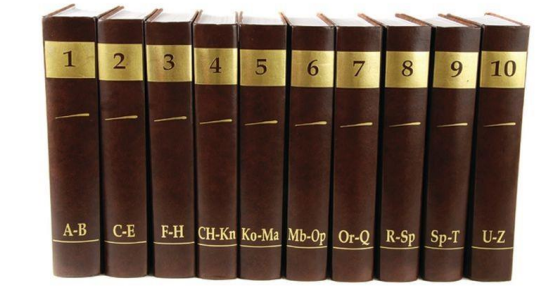


List interfeisas

Sąrašo kolekcijos naudojamos tuomet kai reikia įsiminti elementų tvarką.

Dvi sąrašo realizacijos:

- ArrayList
- LinkedList



Sunumeruotų knygų sąrašas

Set interface

Set yra nesurūšiuota kolekcija unikalių elementų.

Visi elementai yra surūšiuoti, todėl pačių elementų paieška ir įdėjimas, paėmimas ir trynimas yra daug greitesnis.

Dvi konkrečios klasės realizuojančios šią kolekcija:

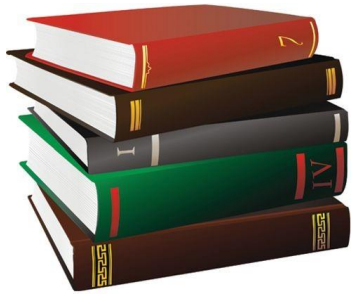
- hash tables
- binary search trees

Knygų krūva (visos unikalios)

Stekas

Prisimena įterpimo tvarką

Galima paimti ir pridėti elementus tik iš viršaus



Knygų krūva

Queue kolekcija

Skirta įsiminti įterptų elementų tvarkai,

Elementai gali būti įterpti į eilės galą arba pradžią

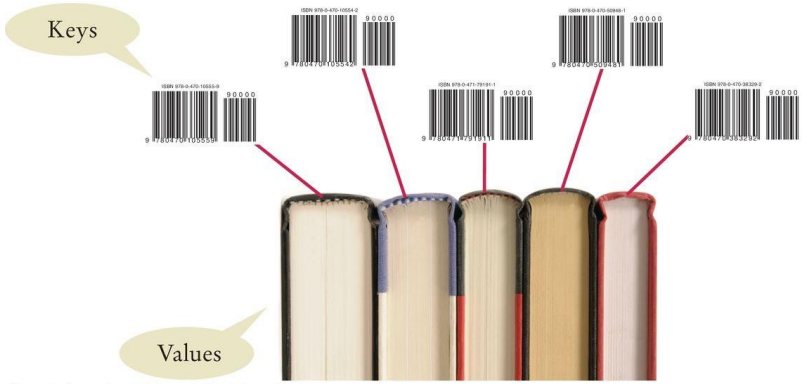
Prioritezeuota eilė gali būti panaudota kaip stekas (ralizuota dvigubu sąrašu)

Photodisc/Punchstock.

Eilė

Map

Išlaiko susietumą tarp reikšmės ir rakto. Raktas gali būti bet koks objektas.



(books) © david franklin/iStockphoto.

A Map from Bar Codes to Books

Skirtumai tarp ArrayList ir Linkedlist

ArrayList

1) ArrayList internally uses **array** to store the elements.

2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.

3) ArrayList class can **act as a list** only because it implements List only.

4) ArrayList is **better for storing and accessing** data.

LinkedList

LinkedList internally uses **doubly linked list** to store the elements.

Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory.

LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.

LinkedList is **better for manipulating** data.

Linked Lists

A data structure used for collecting a sequence of objects:

Allows efficient addition and removal of elements in the middle of the sequence.

A linked list consists of a number of nodes;

Each node has a reference to the next node.

A node is an object that stores an element and references to the neighboring nodes.

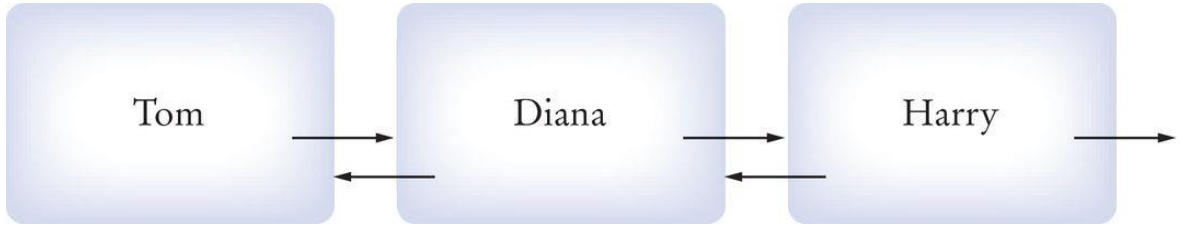
Each node in a linked list is connected to the neighboring nodes.

© andrea laurita/iStockphoto.

Linked Lists

Adding and removing elements in the middle of a linked list is efficient.

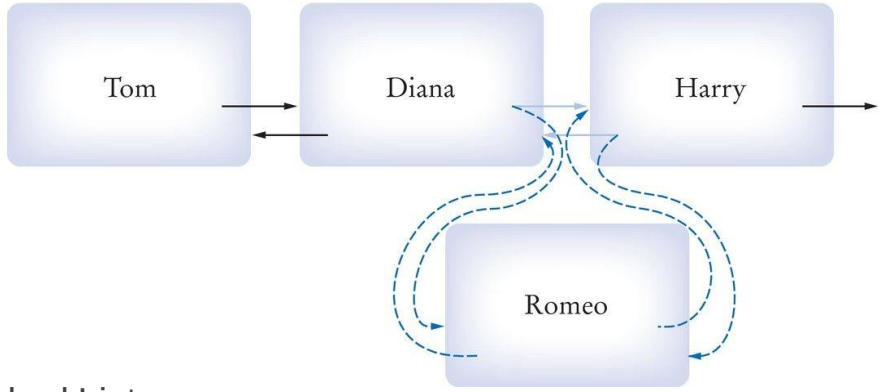
Visiting the elements of a linked list in sequential order is efficient. Random access is **not** efficient.



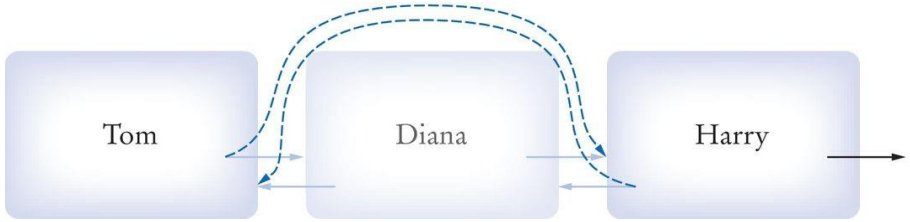
Linked Lists

When inserting or removing a node:

Only the neighboring node references need to be updated



Inserting a Node into a Linked List



Removing a Node From A Linked List

Linkedlist metodai

Keletas pagrindinių LinkedList metodų:

Table 2 Working with Linked Lists	
`LinkedList<String> list = new LinkedList<>();`	An empty list.
`list.addLast("Harry");`	Adds an element to the end of the list. Same as `add`.
`list.addFirst("Sally");`	Adds an element to the beginning of the list. `list` is now `[Sally, Harry]`.
`list.getFirst();`	Gets the element stored at the beginning of the list; here `"Sally"`.
`list.getLast();`	Gets the element stored at the end of the list; here `"Harry"`.
`String removed = list.removeFirst();`	Removes the first element of the list and returns it. `removed` is `"Sally"` and `list` is `[Harry]`. Use `removeLast` to remove the last element.
`ListIterator<String> iter = list.listIterator()`	Provides an iterator for visiting all list elements (see Table 3 on page 684).

List iteratorius

Use a list iterator to access elements inside a linked list. Encapsulates a position anywhere inside the linked list.

To get a list iterator, use the listIterator method of the LinkedList class.

```
LinkedList<String> employeeNames = . . .;

ListIterator<String> iterator = employeeNames.listIterator();
```

List Iterator

Initially points before the first element.

Move the position with next method:

```
if (iterator.hasNext())
{
    iterator.next();
}
```

The next method returns the element that the iterator is passing.

The return type of the next method matches the list iterator's type parameter.

List Iterator

To traverse all elements in a linked list of strings:

```
while (iterator.hasNext())
{
String name = iterator.next();
Do something with name
}
```

To use the “for each” loop:

```
for (String name : employeeNames)
{
Do something with name
}
```

List iteratorius

The nodes of the LinkedList class store two links:

One to the next element

One to the previous one

Called a doubly-linked list

To move the list position backwards, use:

hasPrevious

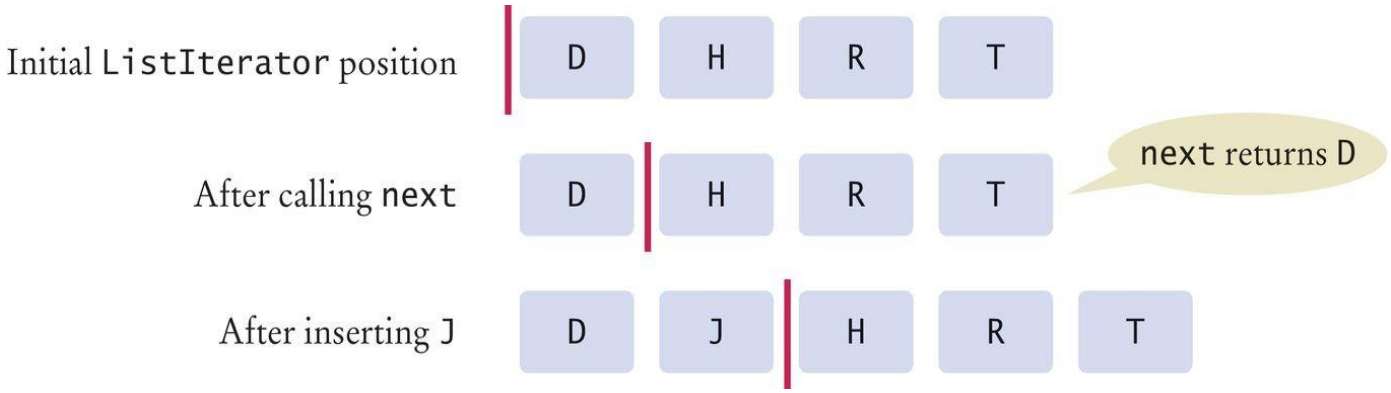
previous

A List Iterator

The add method adds an object after the iterator.

Then moves the iterator position past the new element.

iterator.add("J");



List Iterator šalinimo metodai

Removes object that was returned by the last call to next or previous

To remove all names that fulfill a certain condition:

```
while (iterator.hasNext())
{
        String name = iterator.next();
        if (condition is fulfilled for name)  iterator.remove();
}
```

Be careful when calling remove:

It can be called only once after calling next or previous

You cannot call it immediately after a call to add

If you call it improperly, it throws an IllegalStateException

Iteratoriaus metodai

Keletas pagrindinių iteratoriaus metodų:

Table 3 Methods of the `Iterator` and `ListIterator` Interfaces

`String s = iter.next();`	Assume that `iter` points to the beginning of the list `[Sally]` before calling `next`. After the call, `s` is `"Sally"` and the iterator points to the end.
`iter.previous();` `iter.set("Juliet");`	The `set` method updates the last element returned by `next` or `previous`. The list is now `[Juliet]`.
`iter.hasNext()`	Returns `false` because the iterator is at the end of the collection.

`if (iter.hasPrevious())`
`{`
`  s = iter.previous();`
`}`
	`hasPrevious` returns `true` because the iterator is not at the beginning of the list. `previous` and `hasPrevious` are `ListIterator` methods.

`iter.add("Diana");`	Adds an element before the iterator position (`ListIterator` only). The list is now `[Diana, Juliet]`.
`iter.next();` `iter.remove();`	`remove` removes the last element returned by `next` or `previous`. The list is now `[Diana]`.

LinkedList ir iteratoriaus pavyzdys

```
LinkedList<String> linkedlist = new LinkedList<String>();
linkedlist.add("Pirmas");
linkedlist.add("Antras");

Iterator<String> iteratorius= linkedlist.iterator();

while (iteratorius.hasNext()) {
 System.out.println(iteratorius.next());
}
```

Atvirkštinis duomenų spausdinimas

```
LinkedList<String> linkedlist = new LinkedList<String>();
linkedlist.add("Pirmas");
linkedlist.add("Antras");
linkedlist.add("Trecias");
Iterator<String> iteratorius= linkedlist.descendingIterator();

while (iteratorius.hasNext()) {
 System.out.println(iteratorius.next());
}
```