Свързан списък. Стек. Опашка

Стандартни библиотеки с реализации
на абстрактни типове данни.

.

Annie Ushanova

Въведение

- Масиви (*едномерни и многомерни*) - структури данни с фиксиран размер по време на изпълнение на програмата
- Динамични структури данни- размерът им може да расте или да намалява по време на изпълнение на програмата.

Примери- Абстрактни Типове Данни (АТД)

- Линейни структури
  - Свързани списъци
  - Стекови структури
  - Опашки
- Двоични дървета и графи

Въведение

- **Свързаните списъци** са последователност от данни “свързани във верига”, позволяващи вмъкване и изтриването им на всяко място в списъка.
- **Стек** структурите се използват в компилатори и операционните системи- вмъкване и изтриване може да става само в единия край- върхът на стека.

Въведение

- **Опашки** се използват за представяне на последователност от обекти, изчакващи да изпълнят някакво действие или да бъдат обработени- вмъкване се изпълнява само от края на опашката, а изтриване се извършва от началото на опашката.
  При нея елементите се редят на опашка – първи влязъл, първи излязъл-**FIFO**(First Input First Output)
- **Двоични дървета** спомагат да се търсят и сортират данни с *висока скорост, елиминиране на дублирани стойности*- файлови директории, преобразуване на символни в аритметични изрази и пр.

Абстрактни типове данни

Списък- наредена линейна последователност от данни, чиито стойности могат да се дублират.

Пример: списък от имена, списък от тел. номера

Операции с елементите на списъка:

- Конструктор по подразбиране и копиране;
- Метод за проверка дали списъка е празен;
- Методи за добавяне, изтриване на данна от списъка
- Методи за намиране на първия и последния елемент
- Метод, връщащ стринг с данните от списъка

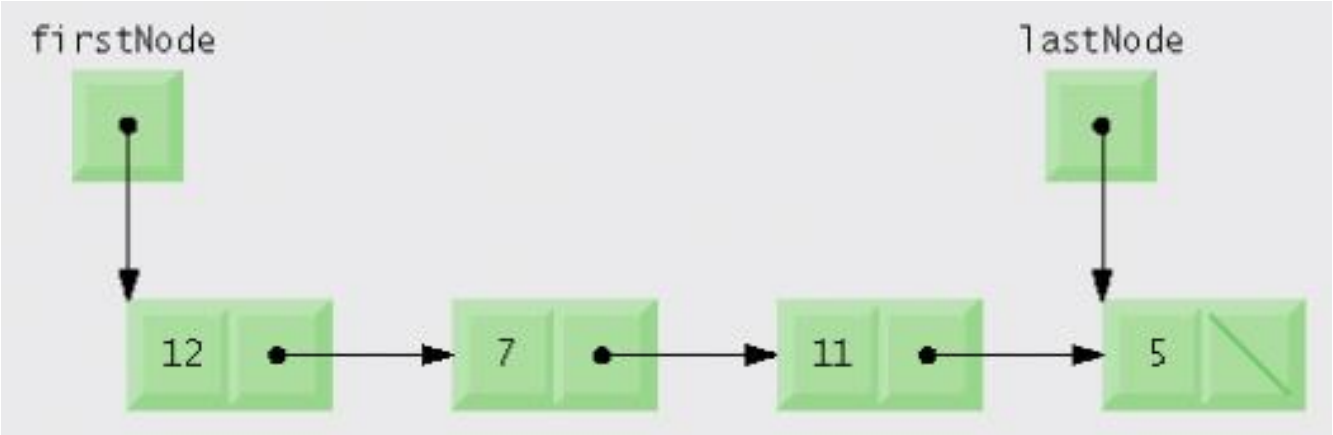
Реализация:

- Данните са елементи на масив
- Данните са във възли на свързан списък. Всеки възел освен данни има референция към следващия и(или) предходния възел

Видове- еднопосочен, двупосочен списък, цикличен списък

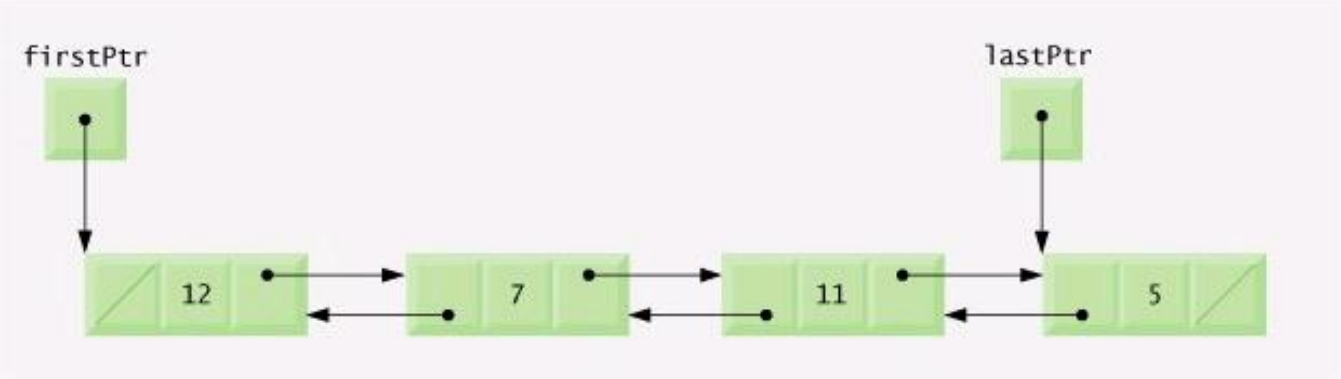
Свързан списък(LinkedList)

- Както подсказва името на тази структура, списъкът представлява **множество от елементи**, **произволно разположени в паметта**, където всеки елемент съхранява в себе си **адреса на следващия елемент**.
- ***Свързано представяне с една връзка***

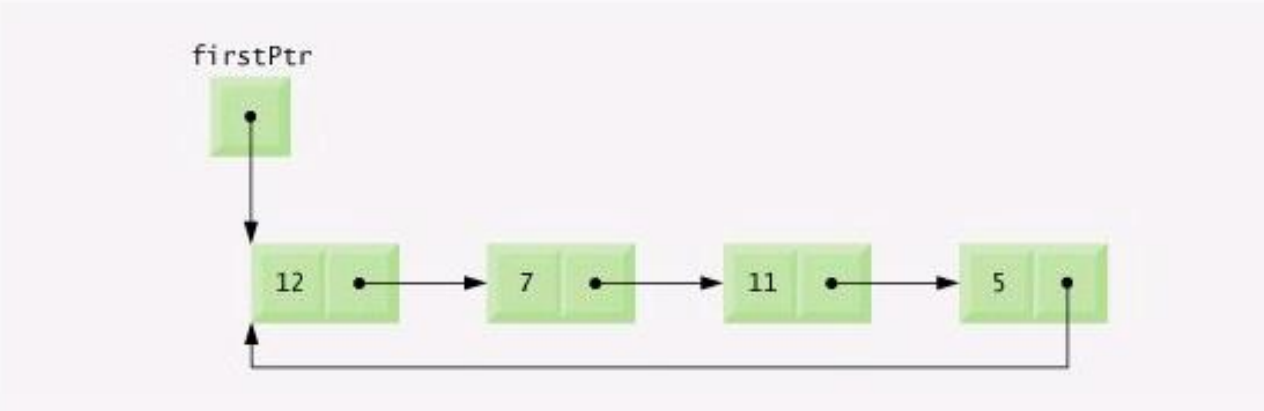


Списък

- ***Свързано представяне с две връзки***

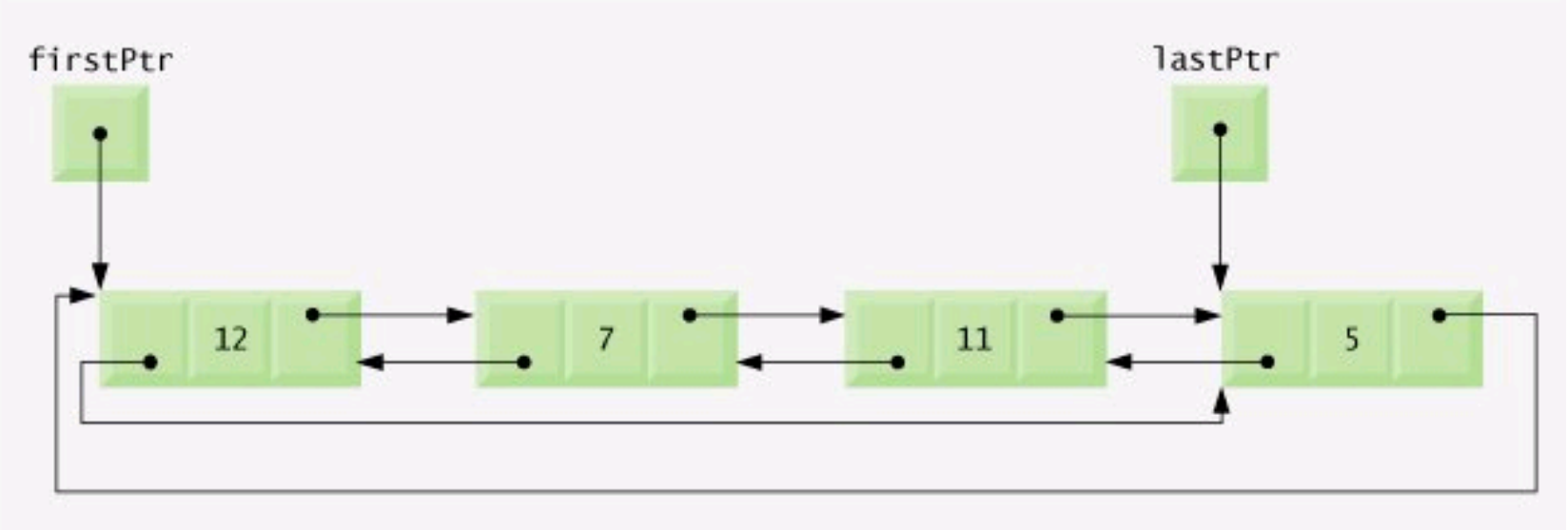


- ***Циклично свързано представяне***



Списък

- ***Затворено свързано представяне с две връзки***



Предимствата и недостатъците на свързания списък спрямо масива?

- **Предимства:**
  - Дава възможност за добавяне или изтриване на елемент.
  - Не е необходимо наличие на голям блок от последователна свободна памет.
- **Недостатъци:**
  - По – бавни са.
  - Нямат директен достъп до елемент
  - Използват повече памет за съхранението на адреси.

Стек

Стек- *линейна колекция от данни с две основни операции- операция за добавяне (`push`) и операция премахване (`pop`) на данна във върха на стека (**LIFO** структура)*.

Пример: стек от карти за игра, стек от кутии, стек от писма и пр.

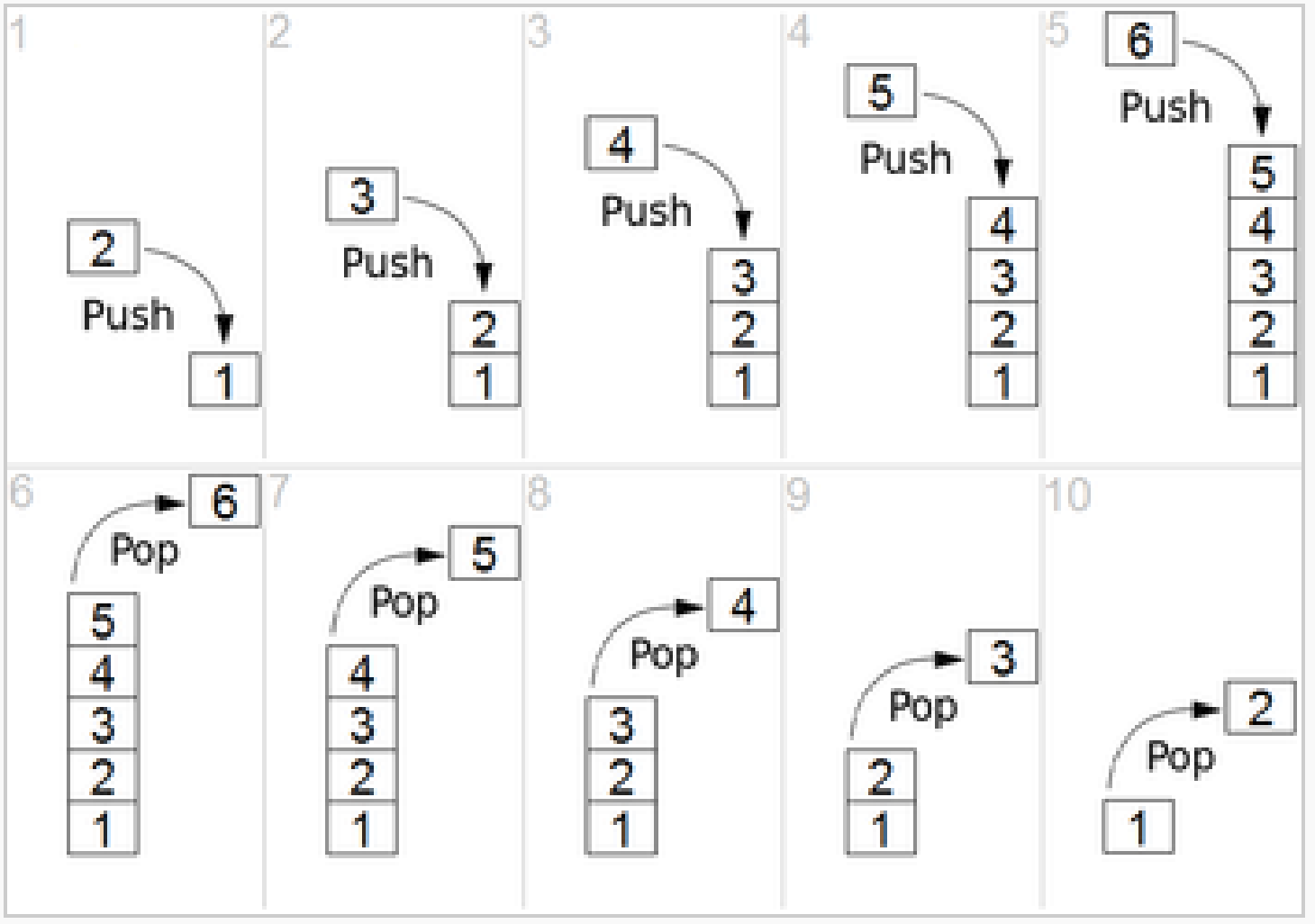
Операции с елементите на стека:

- Конструктор по подразбиране и копиране;
- Метод за проверка дали стека е празен;
- Методи `push и pop`
- Метод за разглеждане (`peek`) на данната във върха на стека
- Метод, връщащ стринг с данните в стека

Реализация:

- Данните са елементи на масив
- Данните са във възли на свързан списък.

Стек STACK FILO



1.2 Абстрактни типове данни

Опашка- *линейна колекция от данни с две основни операции- операция за добавяне (*`enqueue или offer`*) на данна в края на опашката и операция премахване (*`dequeue или poll`*) на данна в началото на опашката (FIFO структура).*

Пример: опашка от задания за принтер, опашка от клиенти и пр.

Операции с елементите на стека:

- Конструктор по подразбиране и копиране
- Метод за проверка дали опашката е празна
- Метод `offer`
- Метод `poll`
- Метод, връщащ стринг с данните в стека

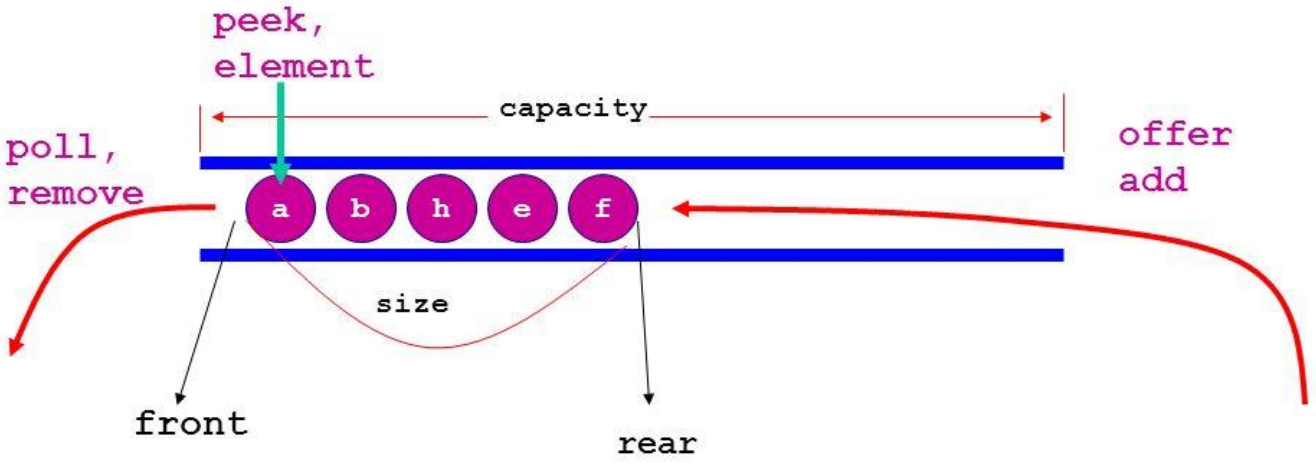
Реализация:

- Данните са елементи на масив,
- Данните са във възли на свързан списък.

Видове- опашка без приоритет и опашка с приоритет

Опашка QUEUE FILO

Queue **FIFO (First In First Out)**



1.3 Колекция от АТД

Java collections framework (`package java.util`)

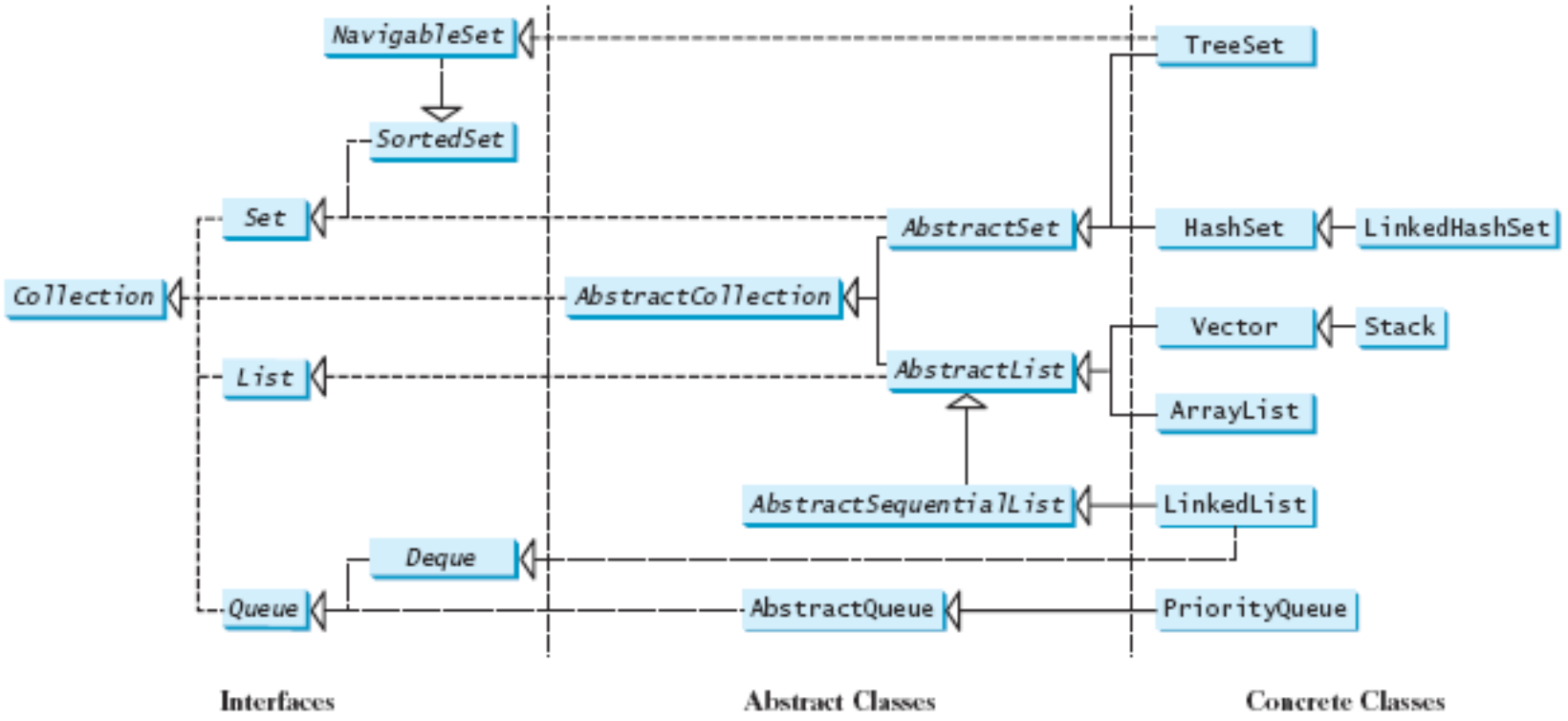
- Съдържа базисни библиотечни реализации на класически структури от данни, интерфейси и съответни алгоритми
- Използва параметри за тип (*generics*)
- Реализации на структурите от данни изучавани дотук
  - Позволява повторно използване на код
  - Позволява повторно използване на готови компоненти
- Съвкупността от базисни компоненти на библиотеката позволява да се използват класическите структури от данни без да се губи време по програмиране на тези компоненти

Колекция от АТД

Базисна система от колекции на АТД (Collections framework)

- Съдържа интерфейси, декларащи операции за различните типове колекции
- Дават високо качество на изпълнение за класическите структури от данни
- Позволяват многократна употреба
- Обогатени възможности за реализация с използване на параметри за тип
- Предоставя стандартни методи за мигриране на данни от една структура данни в друга
- Позволява използване на Ламбда изрази и Stream API

Йерархия на наследственост в базисната библиотека от колекции на Java



List

- Java.util.List е производен интерфейс на Collection.
- Това е подредена колекция от обекти, в която могат да се съхраняват дублиращи се стойности.
- Тъй като List **запазва реда на вмъкване**, той **позволява позиционен достъп** и **вмъкване** на елементи.
- Интерфейсът на List се осъществява от класовете ArrayList, LinkedList, Vector и Stack.

Създаване на List Обекти:

- List е интерфейс и инстанциите на List могат да бъдат създадени чрез имплементиране на различни класове по следните начини:

```
List a = new ArrayList();
List b = new LinkedList();
List c = new Vector();
List d = new Stack();
```

Операции в List:

- Интерфейсът на списъка наследява Collection, следователно той поддържа всички операции на Collection Interface, заедно със следните допълнителни операции:
- **Позиционен достъп (Positional Access:)**
  - Списъкът позволява **добавяне**, **премахване** на елементи, **get** и **set** **операции въз основа на позициятя на елементи** в списъка.

Операции в List (1) :

- **void add(int index, Object O):**
  - Този метод добавя зададен елемент в определен индекс
- **boolean addAll(int index, Collection c):**
  - Този метод добавя към списъка всички елементи от определената колекция. Първият елемент се вмъква на зададения индекс. Ако вече има елемент в тази позиция, този елемент и други последващи елементи (ако има такива) се преместват надясно чрез увеличаване на индекса им

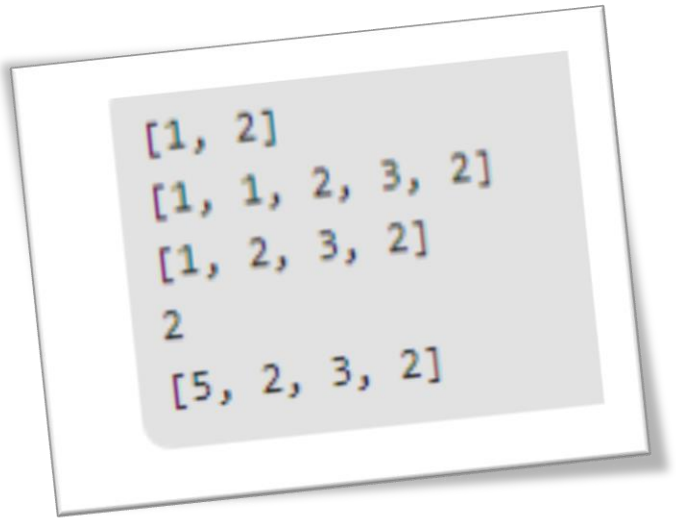
Операции в List (2) :

- **Object remove(int index):**
  - Този метод премахва елемент от указания индекс. Той измества следващите елементи (ако има такива) наляво и намалява индексите им с 1.
- **Object get(int index):**
  - Този метод връща елемент с посочения индекс
- **Object set(int index, Object new):**
  - Този метод заменя елемента в даден индекс с нов елемент. Тази функция връща елемента, който току-що е заменен от нов елемент.

Пример 1

```java
// Java program to demonstrate positional access
// operations on List interface
import java.util.*;

public class ListDemo {
    public static void main(String[] args)
    {
        // Creating a list
        List<Integer> l1 = new ArrayList<Integer>();
        l1.add(0, 1); // adds 1 at 0 index
        l1.add(1, 2); // adds 2 at 1 index
        System.out.println(l1); // [1, 2]

        // Creating another list
        List<Integer> l2 = new ArrayList<Integer>();
        l2.add(1);
        l2.add(2);
        l2.add(3);

        // Will add list l2 from 1 index
        l1.addAll(1, l2);
        System.out.println(l1);

        // Removes element from index 1
        l1.remove(1);
        System.out.println(l1); // [1, 2, 3, 2]

        // Prints element at index 3
        System.out.println(l1.get(3));

        // Replace 0th element with 5
        l1.set(0, 5);
        System.out.println(l1);
    }
}
```

Примерът демонстрира позиционният достъп до елементите на List



Методи за търсене на елемент и връща неговата позиция

- **SEARCH:**
- List предоставя методи за търсене на елемент и връща неговата позиция.
- Следните два метода се поддържат от List за тази операция:
- **int indexOf(Object o):**
  - Този метод **връща първо срещане** на даден елемент или -1, ако елемент не присъства в списъка.
- **int lastIndexOf(Object o):**
  - Този метод **връща последното срещане** на даден елемент или -1, ако елементът не присъства в списъка.

Search – Пример2

```java
// Java program to demonstrate search
// operations on List interface

import java.util.*;

public class ListDemo {
    public static void main(String[] args)
    {
        // Type safe array list, stores only string
        List<String> l = new ArrayList<String>(5);
        l.add("СОФИЙСКА");
        l.add("ПРОФЕСИОНАЛНА");
        l.add("ГИМНАЗИЯ");

        // Using indexOf() and lastIndexOf()
        System.out.println("first index:"
                           + l.indexOf("СОФИЙСКА"));
        System.out.println("last index:"
                           + l.lastIndexOf("ГИМНАЗИЯ"));
        System.out.println("Index of element"
                           + " not present : "
                           + l.indexOf("Hello"));
    }
}
```

```
first index:0
last index:2
Index of element not present : -1
```

Range-view:

- List Interface предоставя **метод за получаване на част от даден списък между два индекса**.

- **List subList(int fromIndex, int toIndex):**
- Този метод връща списък, който съдържа елементите между специфичен начален и краен индекс от обработвания списък fromIndex(inclusive) и toIndex(exclusive).

```java
// Java program to demonstrate subList operation
// on List interface.
import java.util.*;
public class ListDemo {
    public static void main(String[] args)
    {
        // Type safe array list, stores only string
        List<String> l = new ArrayList<String>(5);

        l.add("SOFIA");
        l.add("PROFESSIONAL");
        l.add("HIGH SCHOOL ");
        l.add("OF ");
        l.add("ELECTRONICS");

        List<String> range = new ArrayList<String>();

        // Return List between 2nd(including)
        // and 4th element(excluding)
        range = l.subList(2, 4);

        System.out.println(range);
    }
}

```

```
[HIGH SCHOOL , OF ]
```

```java
// Java program to demonstrate subList operation
// on List interface.
import java.util.*;
public class ListDemo {
    public static void main(String[] args)
    {
        // Type safe array list, stores only string
        List<String> l = new ArrayList<String>(5);

        l.add("SOFIA");
        l.add("PROFESSIONAL");
        l.add("HIGH SCHOOL ");
        l.add("OF ");
        l.add("ELECTRONICS");

        List<String> range = new ArrayList<String>();

        // Return List between 2nd(including)
        // and 4th element(excluding)
        range = l.subList(1, 3);

        System.out.println(range);
    }
}
```

```
[PROFESSIONAL, HIGH SCHOOL ]
```