Списки

Составитель: Рощупкин Александр

Структуры данных

«Плохие программисты думают о коде. Хорошие программисты думают о структурах данных и их взаимосвязях», — Линус Торвальдс, создатель Linux

* **Структура данных** — это контейнер, который хранит данные в определенном макете. Этот «макет» позволяет структуре данных быть эффективной в некоторых операциях и неэффективной в других

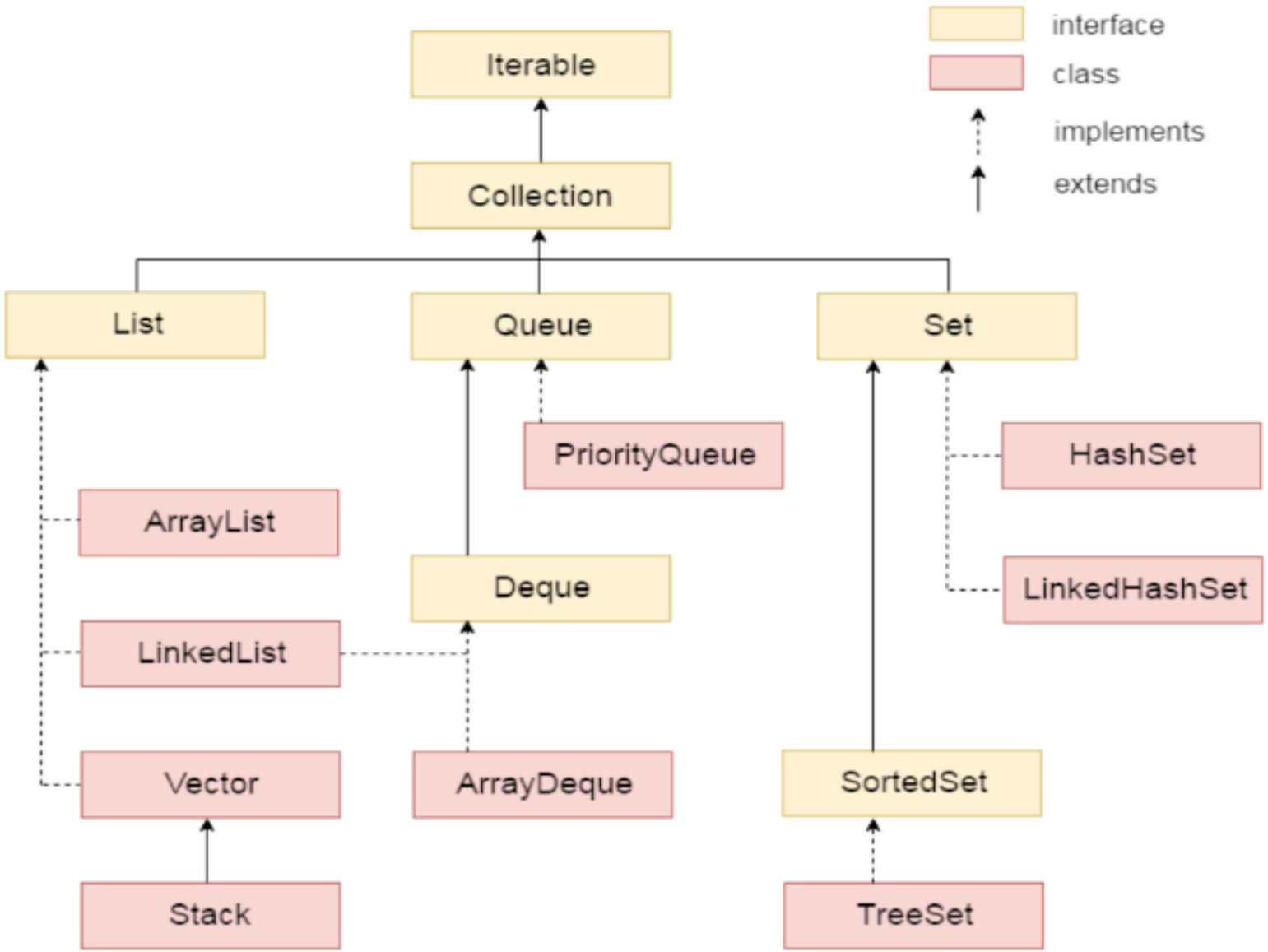
Основные структуры данных

* Массивы
* Множества
* Очереди
* Списки
* Связанные списки
* Графы
* Деревья
* Хэш таблицы

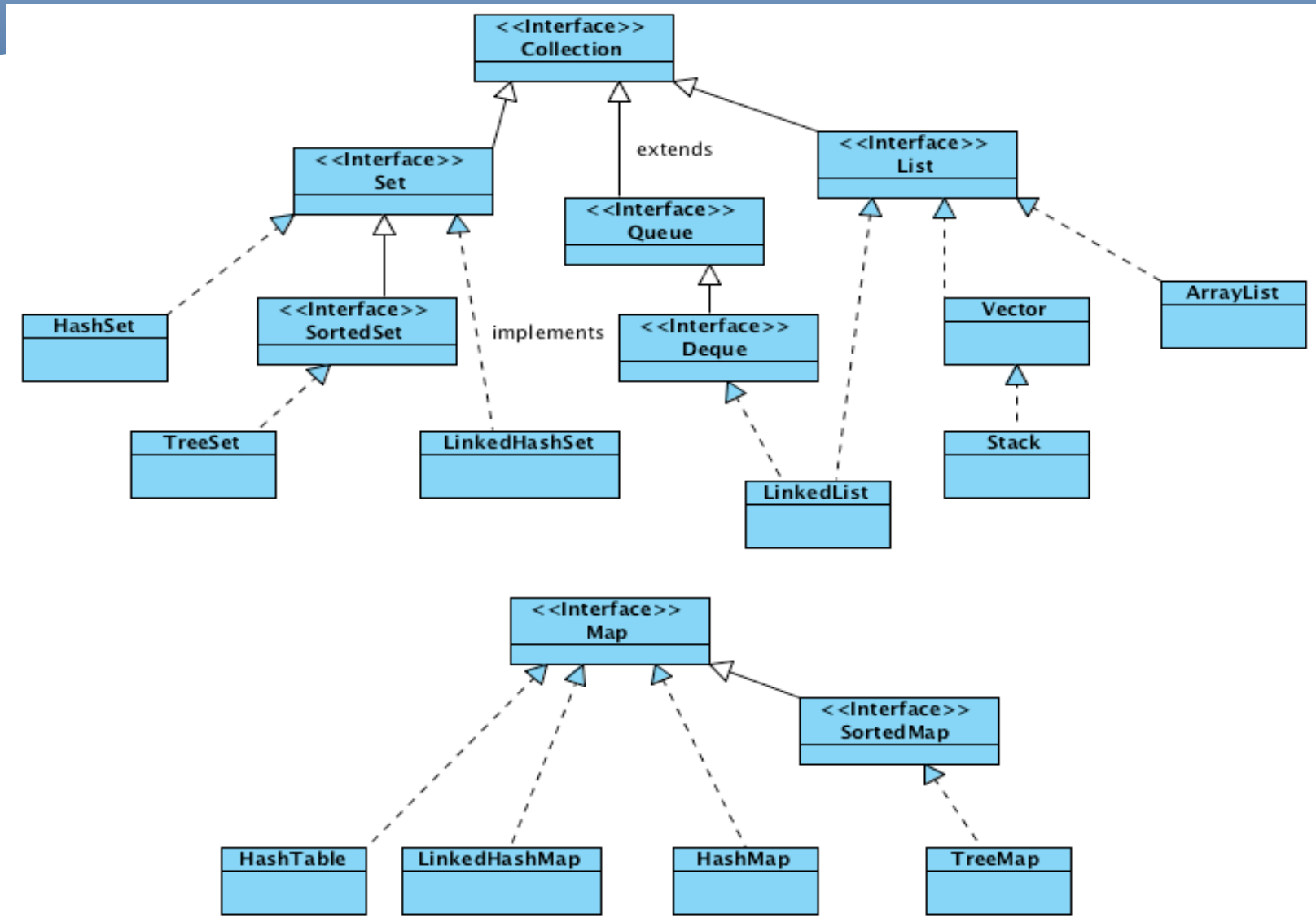
Коллекции

* Коллекции – это подбиблиотека в рамках стандартной библиотеки java, содержащая основные структуры данных и алгоритмы работы с ними

Коллекции


interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

Коллекции и карты



Основные элементы коллекций

* **Collection:** базовый интерфейс для всех коллекций и других интерфейсов коллекций
* **Queue:** наследует интерфейс Collection и представляет функционал для структур данных в виде очереди
* **Deque:** наследует интерфейс Queue и представляет функционал для двунаправленных очередей
* **List:** наследует интерфейс Collection и представляет функциональность простых списков
* **Set:** также расширяет интерфейс Collection и используется для хранения множеств уникальных объектов

Список на основе массива

* Список – структура данных, хранит элементы последовательно
* Каждому элементу данных присваивается положительное числовое значение (индекс), который соответствует позиции элемента в массиве. Начальный индекс массива равен 0

Заготовка списка

```
public class VectorList {
    private int[] vector; // основной массив
    public static final int SIZE = 16; //
размер массива
    private int length = 0; // виртуальная
длинна

    public VectorList() {
        this.vector = new int[SIZE]; //
создание реального массива
    }

    public int get(int index) {
        checkIndex(index); // проверка на
```

Обобщённое программирование

* **Обобщённое программирование** — это такой подход к описанию данных и алгоритмов, который позволяет их использовать с различными типами данных без изменения их описания. Generics (дженерики) или <> — подмножество обобщённого программирования

Обобщения (Дженерики)

* **Дженерики** - это параметризованные типы. С их помощью можно объявлять классы, интерфейсы и методы, где тип данных указан в виде параметра
* Обобщения - добавили в язык java безопасность типов, в основном при использовании коллекций
* При создании объекта параметризированного класса, нужно указать конкретное значение для переменной типа

Пример не защищённого класса

```
class Box {
    private Object value;

    public Box(Object value) {
        this.value = value;
    }

    public Object get() {
        return value;
    }
}
```

Пример защищённого класса

```
class Box<E> {
    private E value;

    public Box(E value) {
        this.value = value;
    }

    public E get() {
        return value;
    }
}
```

Название переменных типа

* В именах переменных типа принято использовать заглавные буквы. Обычно для коллекций используется буква E, буквами K и V - типы ключей и значение (Key/Value), а буквой T (и при необходимости буквы S и U) - любой тип.

Использование параметризированных классов

```
Box<Tea> box1 = new Box<Tea>(new Tea());
Tea tea = box1.get();
Box<Coffee> box2 = new Box<Coffee>(new Coffee());
Coffee tea1 = box2.get();
```

* Алмазный стиль позволяет не задавать значение типа при создании объекта, если его можно вычислить по объявлению ссылочной переменной
* Map<Integer, String> pair = new HashMap<>();

Как это работает

* Поддержка generic-ов реализована средствами компилятора. Виртуальная машина не предоставляет никакой поддержки generic-ов, кроме возможности получения информации о типах
* Во время компиляции generic-параметры убираются, и, там, где это требуется, вместо них вставляется приведение типов.

Несовместимость дженерик типов

* Для того чтобы сохранить целостности и независи мости друг от друга, у Generics существует так наз ываемая "Несовместимость generic-типов»

```
List<Integer> li = new ArrayList<>();
List<Object> lo = li; // ошибка компиляции
lo.add("hello");
```

Ограничения дженериков

* Невозможно создать объект generic типа, поскольку компилятор не знает, какой конструктор вызвать.
* private static <T> T get(T value) { return new T(); }

Ограничения дженериков

Невозможно реализовывать одновременно два один аковых интерфейса с разными типами.

```
public abstract class DecimalString implements
Comparable<Number>, Comparable<String> {}
```

Ограничения дженериков

* Невозможно объявить статическое поле generic типа

```
public class MyClass<T> {
    private static T value;
}
```

Ограничения дженериков

Невозможно использовать instatceof для параметризованного типа

```
public static <E> void setList(List<E> list) {
     if (list instanceof ArrayList) {

     }
}
```

Ограничения дженериков

Невозможно создать массив параметризированного типа

```
Box<Tea>[] arrayOfLists = new Box<>[2]; // compilation error
```

Ограничения дженериков

* Невозможно перегрузить метод, в котором типы параметров “стираются” до одного и того же типа.
* public void print(Set<Integer> strSet) { }
* public void print(Set<String> intSet) { }

Шаблоны аргументов (Wildcards)

* Шаблон аргументов указывается символом ? и представляет собой неизвестный тип

```
Object obj = new Object();
Box<?> box3 = new Box<Object>(obj );
```

Приемущества шаблонов

* Одно из преимуществ wildcards состоит в том, что они дают возможность написать код, который может оперировать различными generiстипами без знания их точных границ

```
public void unbox(Box box) {
    System.out.println(box.get());
}

public void rebox(Box box) {
    box.put(box.get());
}
```

Пример

```
public void rebox(Box<?> box) {
    reboxHelper(box);
}

private<V> void reboxHelper(Box<V> box) {
    box.put(box.get());
}
```

Описание

* Вспомогательный метод reboxHelper() является generic-методом. Generic-методы вводят дополнительные параметры типов (помещаемые в угловые скобки перед типом возвращаемого значения), которые обычно используются для формулирования ограничений типов между параметрами и/или возвращаемым значением метода.

Описание

* Однако в случае reboxHelper() genericметод не задействует параметр типа для определения ограничения типа, а позволяет компилятору – через вывод типа – дать имя параметру типа переменной box. Приём с capturехелпером основан на выводе типов (type inference) и преобразовании при фиксации (capture conversion)

Маски с ограничением extends

```
Box<? extends Coffee> box3 = new Box<Coffee>(new
Coffee());
Box<? extends Tea> box3 = new Box<Tea>(new Tea());
```

Параметризированные методы

* По аналогии с универсальными классами (дженерик-классами), можно создавать универсальные методы (дженерик-методы), то есть методы, для которых можно задать свои дженерики
* Универсальные методы не надо путать с методами в generic-классе. Универсальные методы удобны, когда одна и та же функциональность должна применяться к различным типам

Пример универсального метода

```
class Utilities {
    public static <T> void fill(List<T> list, T val) {
        for (int i = 0; i < list.size(); i++) {
            list.set(i, val);
        }
    }
}
```

Параметризированный список

```
public class VectorListGener<E> {
    private E[] vector; // основной массив
    public static final int SIZE = 16; // размер массива
    private int length = 0; // виртуальная длинна

    public VectorListGener() {
        this.vector = (E[]) new Object[SIZE]; // создание реального массива
    }

    public E get(int index) {
        checkIndex(index); // проверка на выход за границы
        return vector[index];
    }

    private void checkIndex(int index) {
        if (index < 0 || index > length) {
            throw new IndexOutOfBoundsException(String.valueOf(index));
        }
    }
}
```

Интерфейс List

- public String toString()
- public void clear()
- public Object[] toArray()
- public Iterator<E> iterator()
- public boolean contains(Object o)
- public boolean isEmpty()
- public int size()
- public int lastIndexOf(Object o)
- public int indexOf(Object o)
- public boolean remove(Object o)
- public E remove(int index)
- public void add(int index, E element)
- public E set(int index, E element)

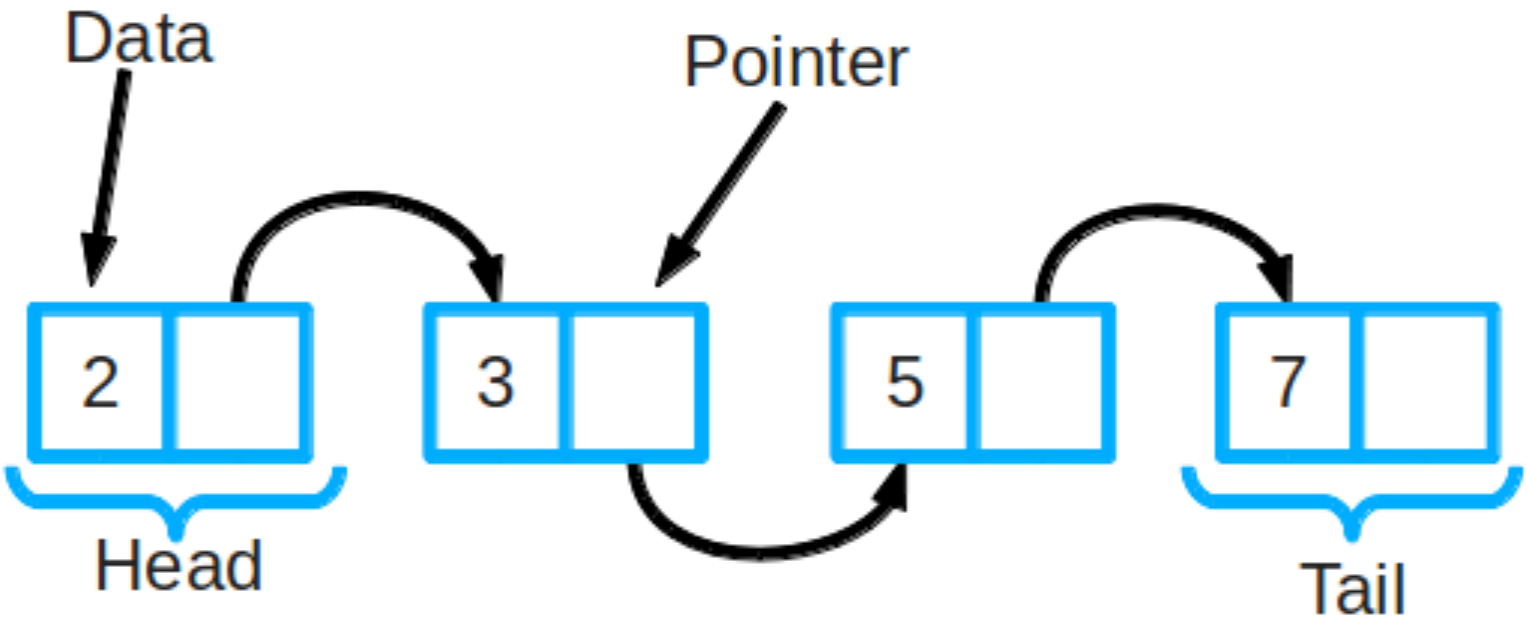
Домашнее задание

- Реализовать перечисленные методы интерфейса List

Связанный список

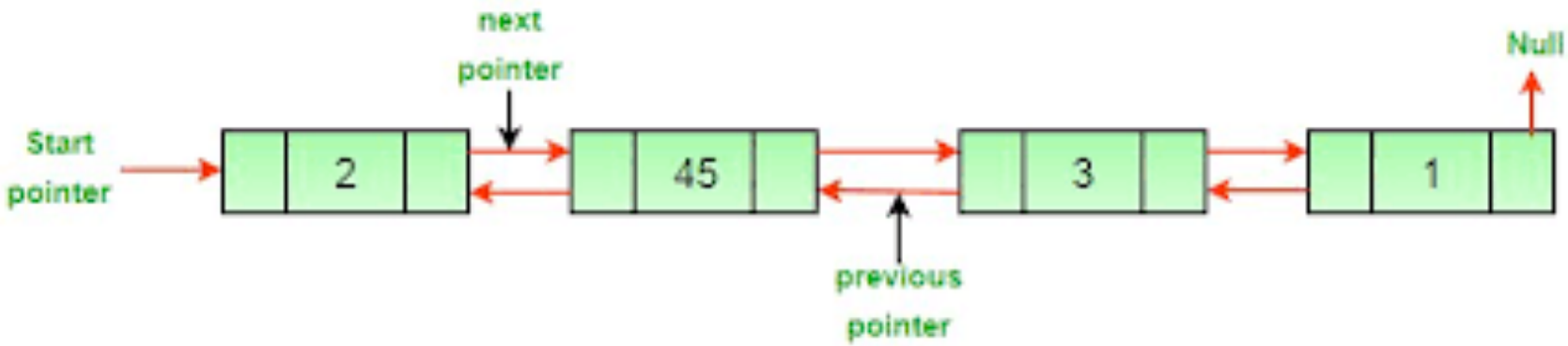
* Связный список состоит из группы узлов, которые вместе образуют последовательность. Каждый узел содержит две вещи: фактические данные, которые в нем хранятся (это могут быть данные любого типа) и указатель (или ссылку) на следующий узел в последовательности. Также существуют двусвязные списки: в них у каждого узла есть указатель и на следующий, и на предыдущий элемент в списке.

Связанный список



Параметризированный связанный список

Двусвязанный список



Домашнее задание

- Релизовать методы интерфейса List, которые реализовывались в задании для списка на основе массива