TECHIN

KOLEKCIJOS (I)

Jaroslav Grablevski

Turinys

- Kolekcijos
- Sąsaja Iterable<E>
- Kolekcijų sąsaja Collection<E>
- Sąsaja List<E>
- ArrayList
- LinkedList
- Collections metodai
- Metodai equals() ir hashcode()
- Sąsaja Comparator<T>
- Sąsaja Comparable<T>

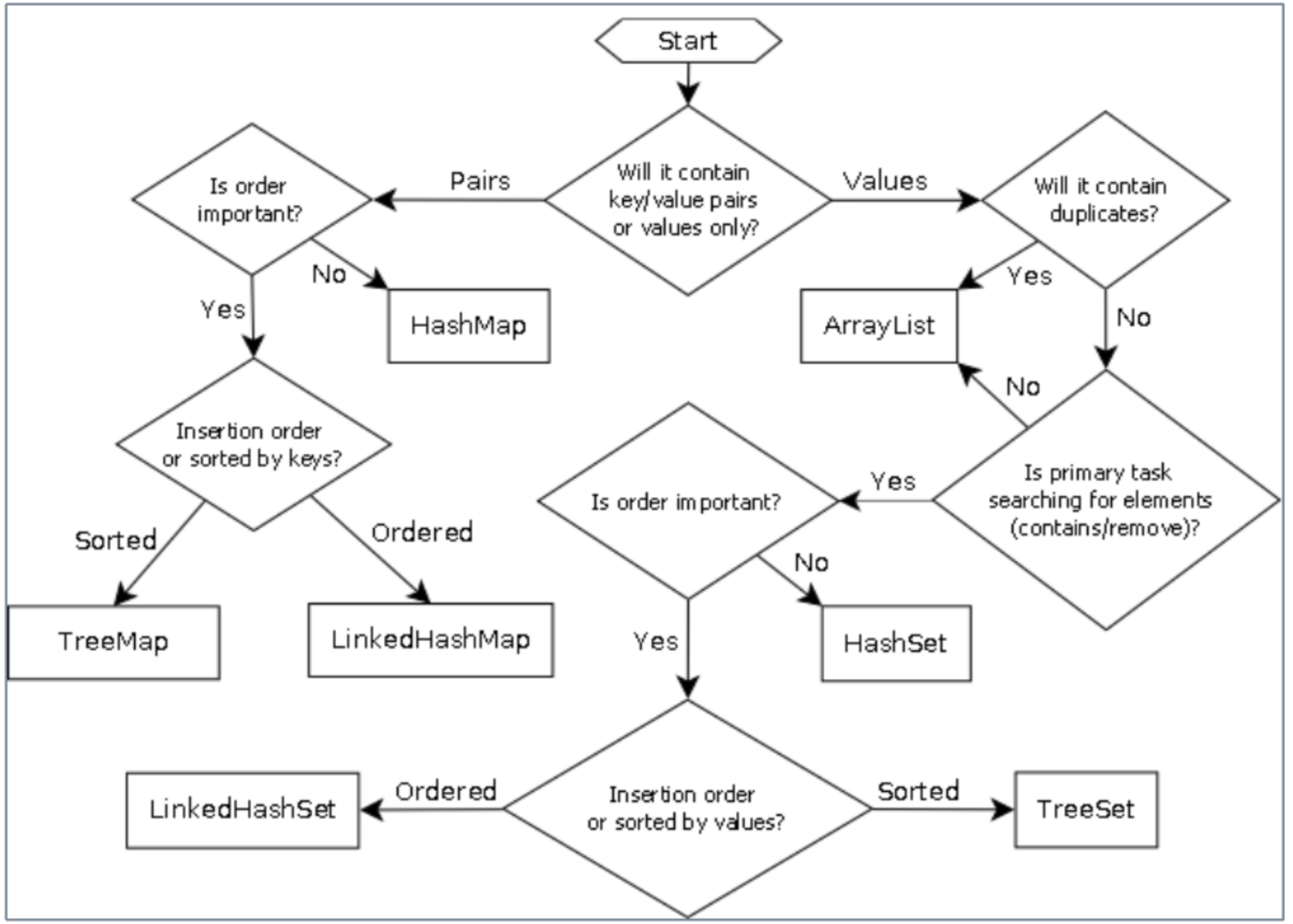
Kolekcijos / Collections

Kolekcijos - objektų saugojimo būdai.

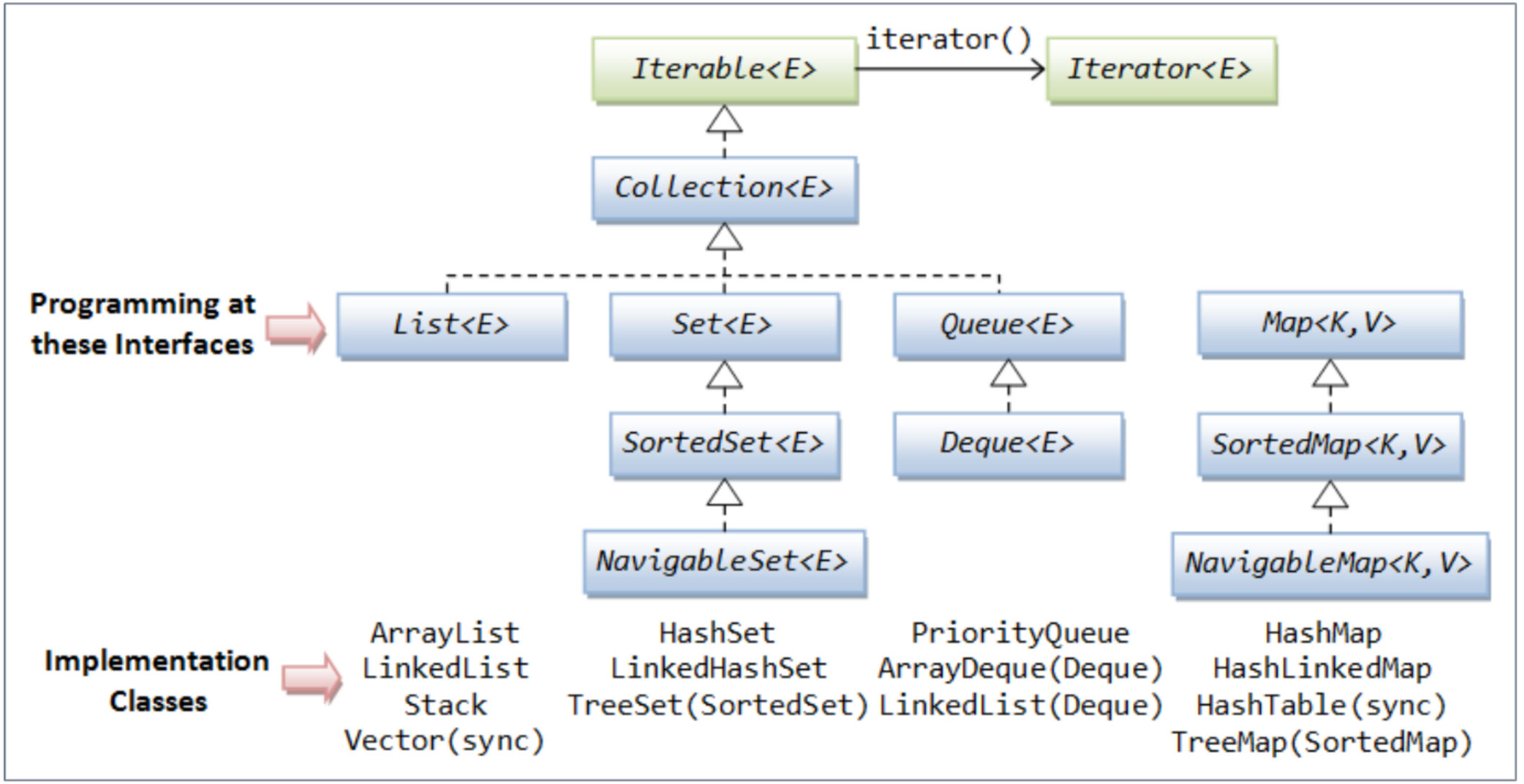
- Java kolekcijų biblioteka pateikia duomenų struktūras bei interfeisus ir algoritmus darbui su šiomis struktūromis
- Kiekvienas objektų organizavimo modelis turi savo privalumų ir trūkumų

Kolekcijos / Collections (1)

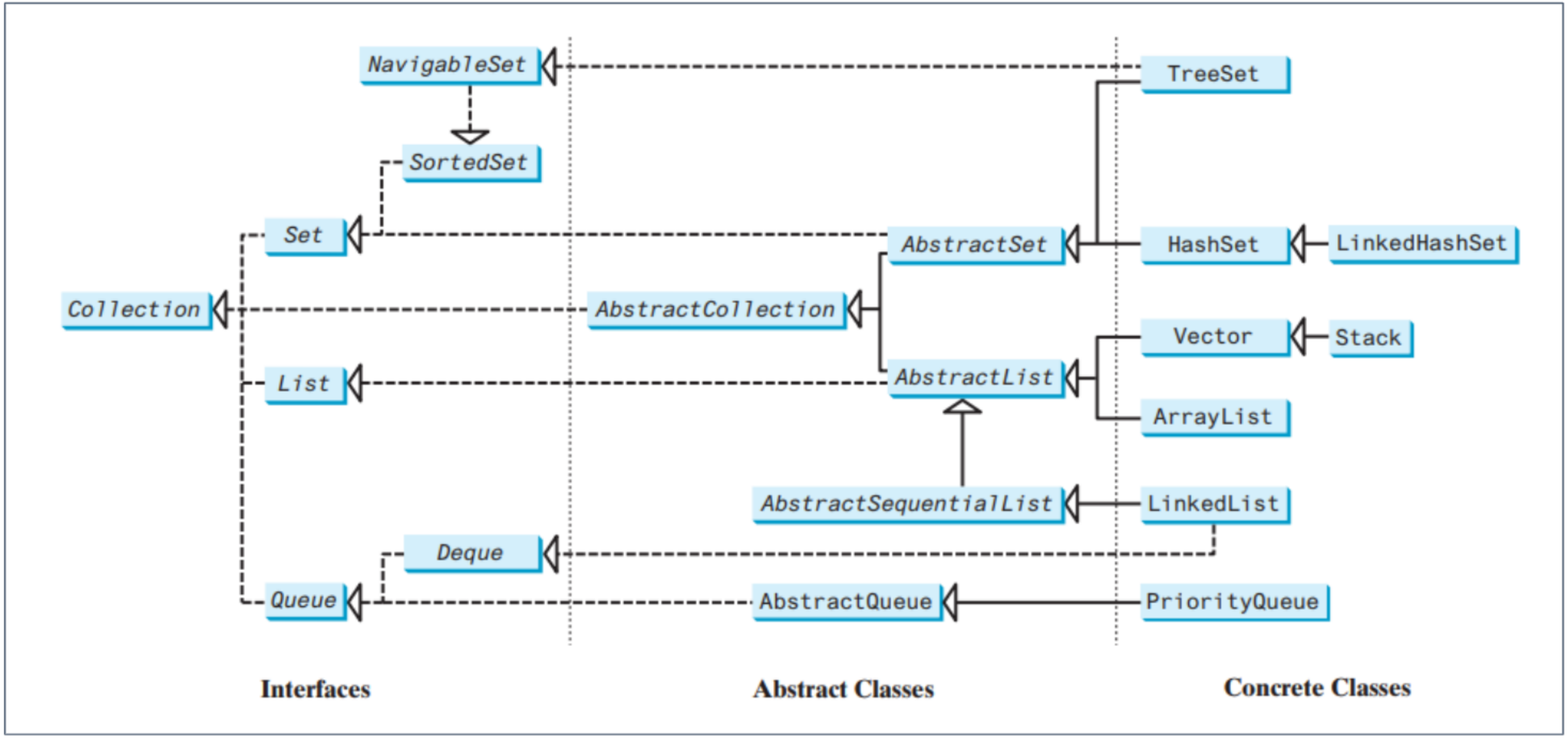
- Svarbu tinkamai parinkti



Sąsajos


Iterable<E>
iterator()
Iterator<E>
Collection<E>
Programming at these Interfaces
List<E>
Set<E>
Queue<E>
Map<K,V>
SortedSet<E>
Deque<E>
SortedMap<K,V>
NavigableSet<E>
NavigableMap<K,V>
Implementation Classes
ArrayList
LinkedList
Stack
Vector(sync)
HashSet
LinkedHashSet
TreeSet(SortedSet)
PriorityQueue
ArrayDeque(Deque)
LinkedList(Deque)
HashMap
HashLinkedMap
HashTable(sync)
TreeMap(SortedMap)

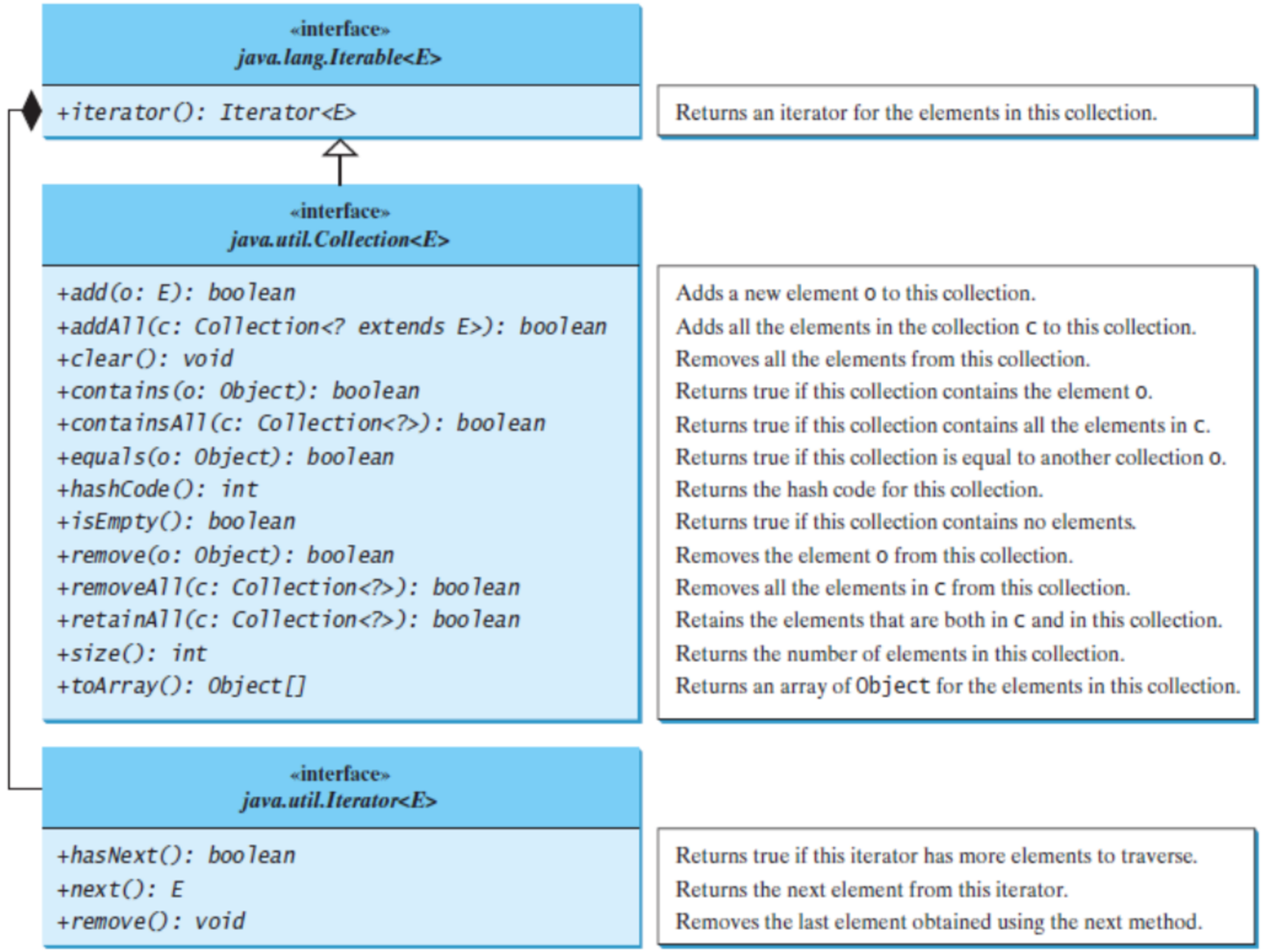
Sąsajos


NavigableSet
SortedSet
Set
Collection
List
Deque
Queue
AbstractSet
AbstractCollection
AbstractList
AbstractSequentialList
AbstractQueue
TreeSet
HashSet
LinkedHashSet
Vector
Stack
ArrayList
LinkedList
PriorityQueue
Interfaces
Abstract Classes
Concrete Classes

Kolekcijų sąsaja Collection<E>

- Sąsaja aprašo papildomus metodus, skirtus darbui su objektais ir pačiu objektu. Juos galima suskirstyti į grupes:
  - Elementų įdėjimą: *add, addAll*;
  - Elementų šalinimą: *remove, clear, removeAll*, *retainAll*;
  - Kolekcijos analizė: *contains, containsAll*, *isEmpty, size*;

Kolekcijų sąsaja Collection<E>



Sąsaja Iterable<E>

- Sąsaja Iterable<E> leidžia naudoti cikle foreach*

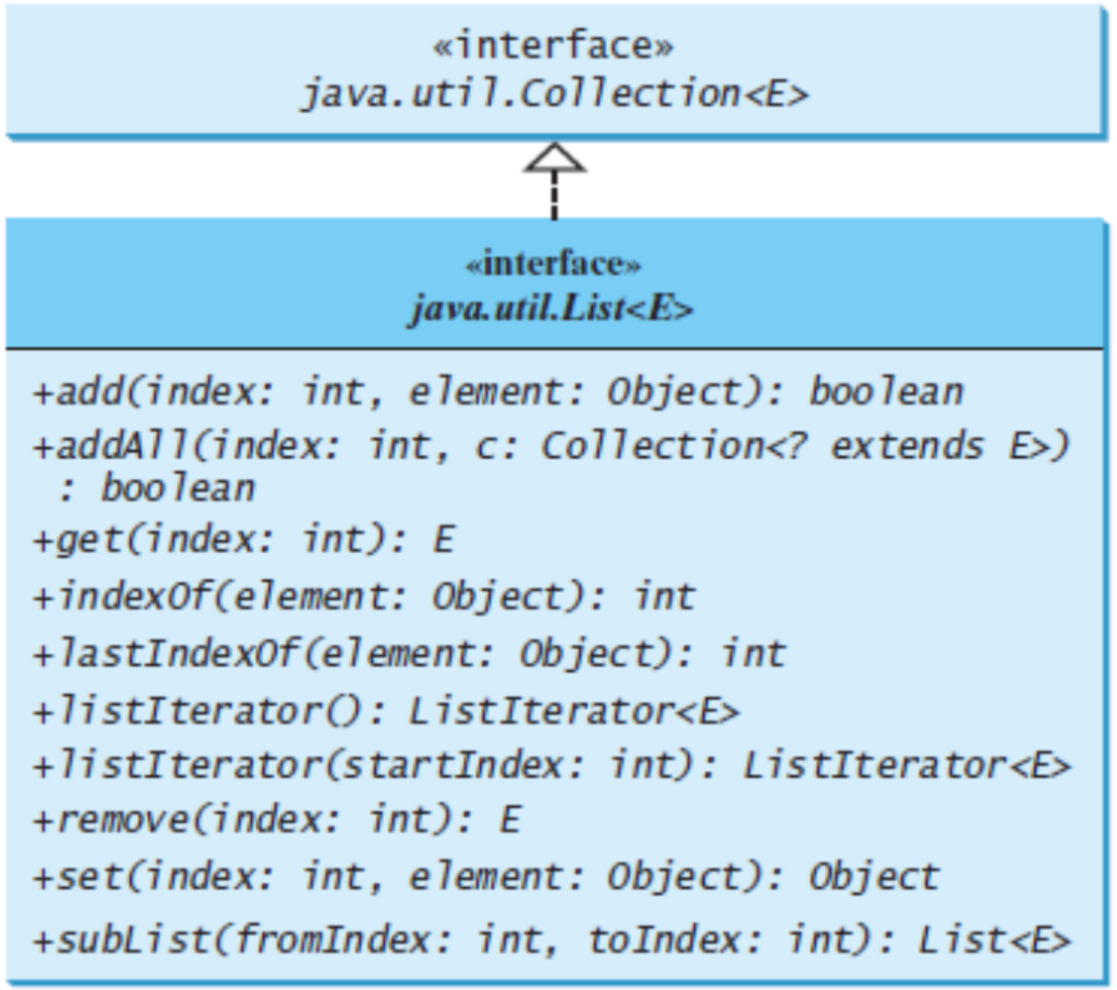
```
Collection<String> collection = new ArrayList<>();
collection.add("a");
collection.add("b");
collection.add("c");

Iterator<String> iterator = collection.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next().toUpperCase() + " ");
}

for (String s : collection) {
    System.out.println(s.toUpperCase() + " ");
}
```

Sąsaja List<E> / The List Interface (1)

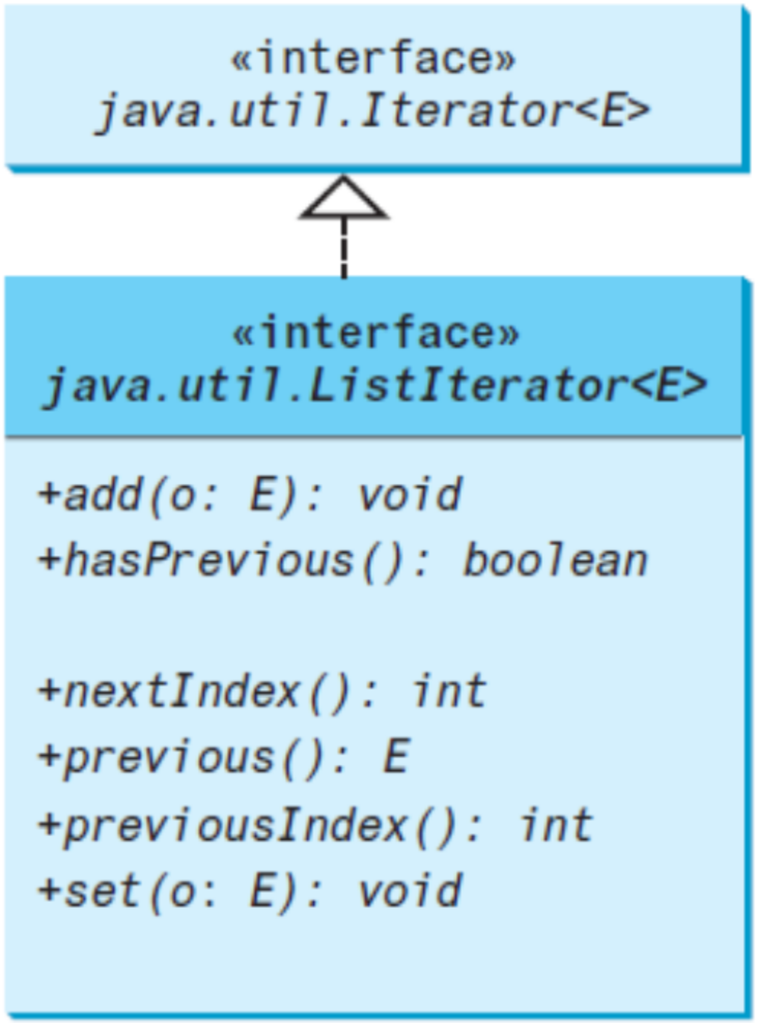
```
public interface List<E> extends Collection<E>
```

- Saugo objektus ta tvarka, kuria jie buvo surašyti;
- Elementus galima tiek dubliuoti, tiek rikiuoti;
- Elementą galima pasiekti tiek indeksu, tiek reikšme;
- Pagrindinės įdiegiančios klasės:
  - ArrayList;
  - LinkedList;

Sąsaja List<E> / The List Interface (1)



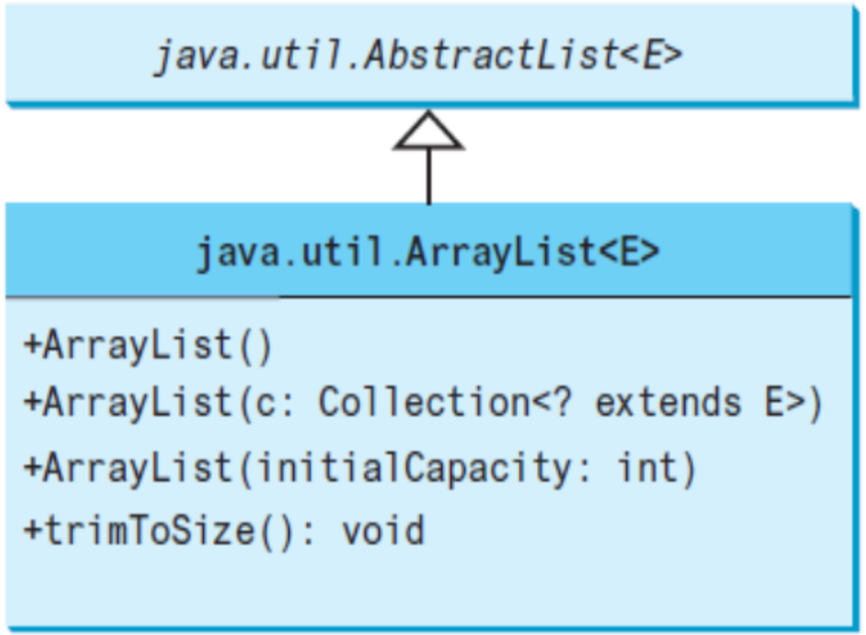
Sąsaja ListIterator<E>



Adds the specified object to the list.
Returns true if this list iterator has more elements when traversing backward.
Returns the index of the next element.
Returns the previous element in this list iterator.
Returns the index of the previous element.
Replaces the last element returned by the `previous` or `next` method with the specified element.

ArrayList<E>

- ArrayList įgyvendina List sąsają naudojant masyvą



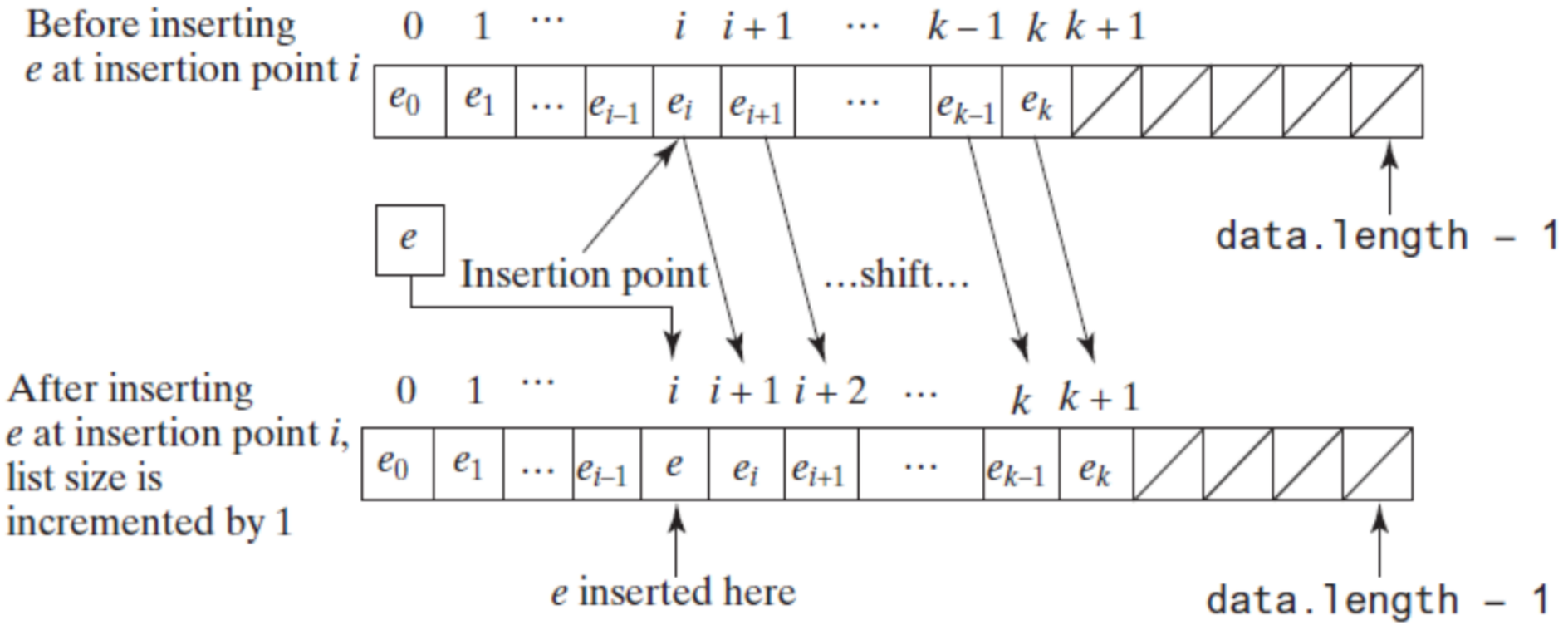
Creates an empty list with the default initial capacity.	
Creates an array list from an existing collection.	
Creates an empty list with the specified initial capacity.	
Trims the capacity of this `ArrayList` instance to be the list's current size.	

ArrayList<E>

```java
ArrayList<Integer> numbers = new ArrayList<Integer>();

// Adding
numbers.add(5);
numbers.add(77);
numbers.add(29);
numbers.add(5);

// Removing
// slow
numbers.remove(0);
// fast
numbers.remove(numbers.size() - 1);
```

ArrayList<E>

- ArrayList elemento įterpimas

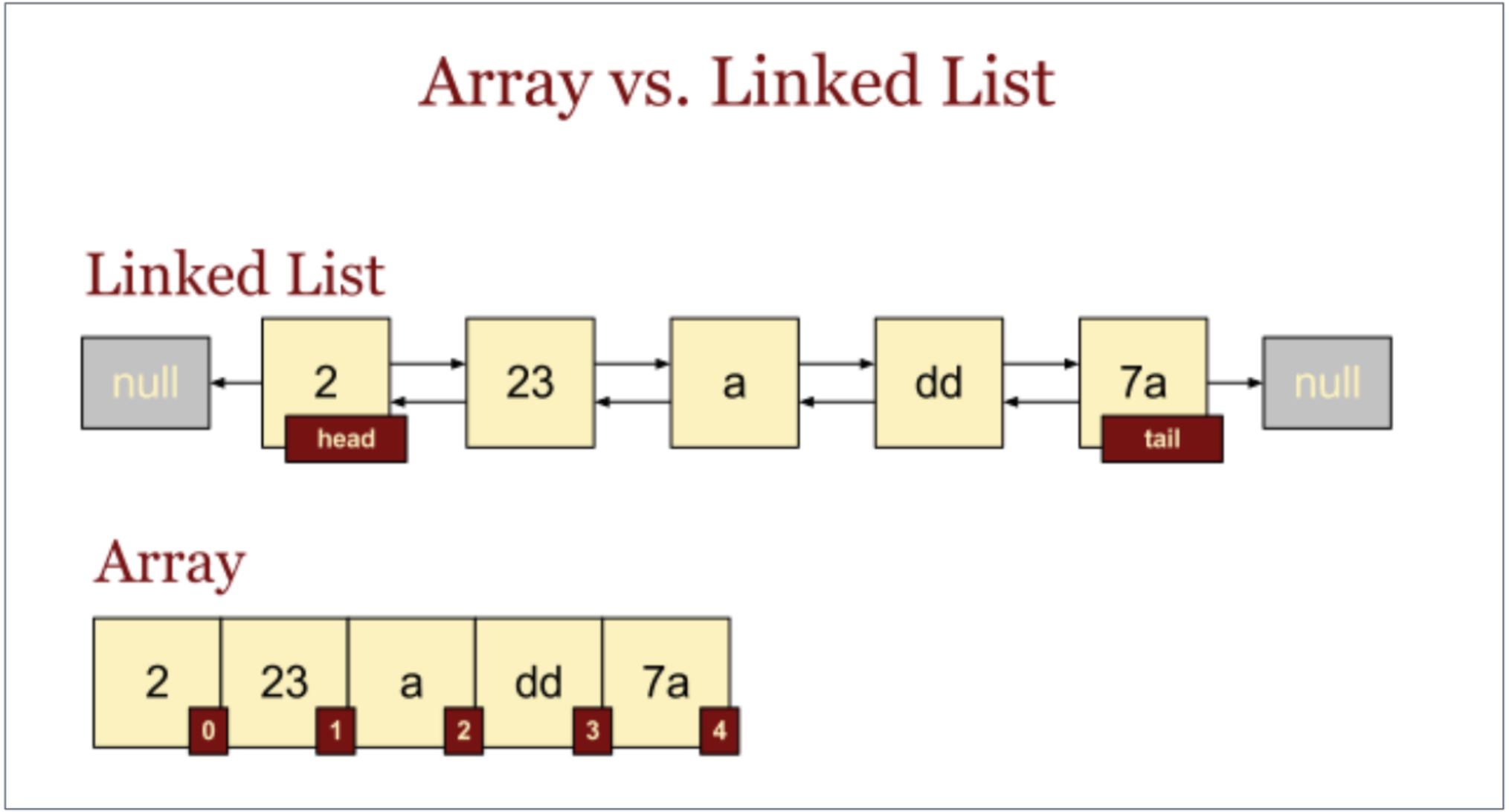


Sąsaja List<E> / The List Interface

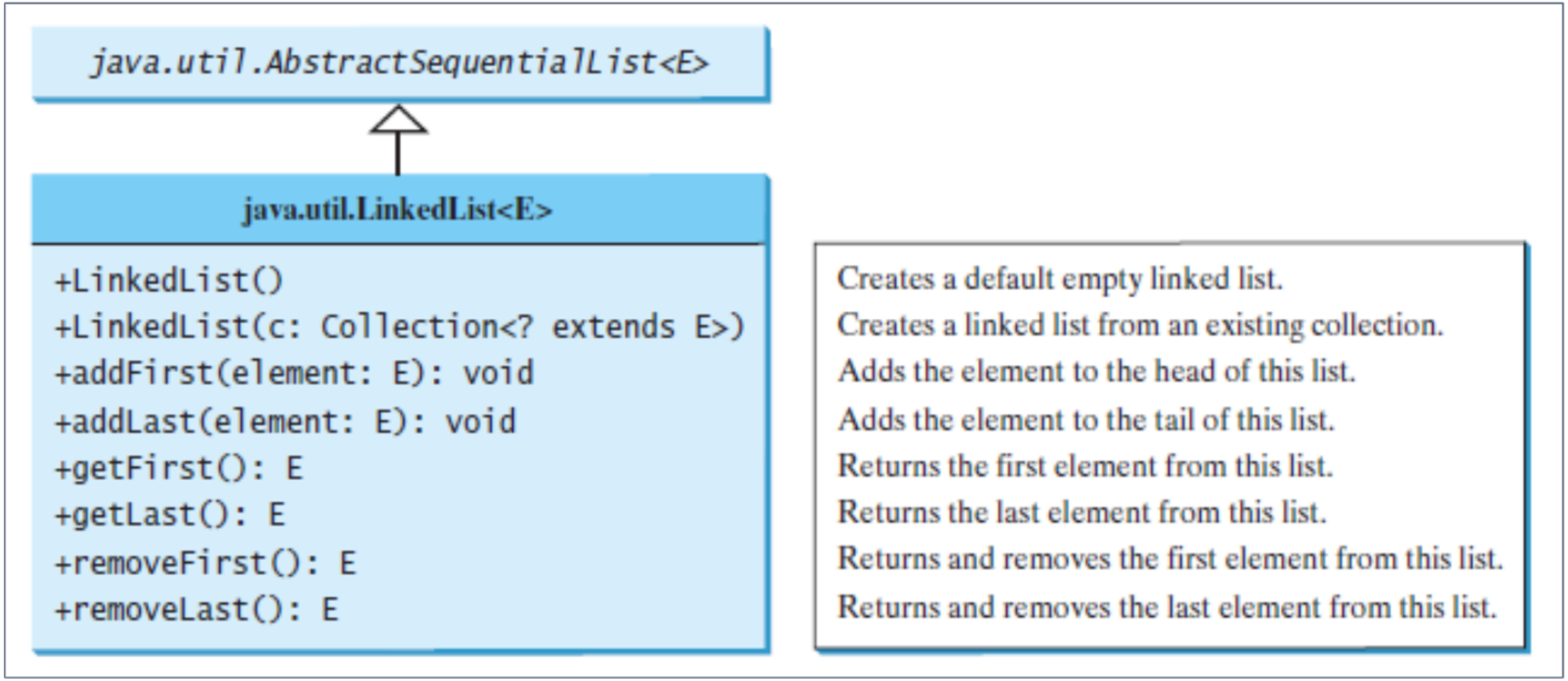
- ArrayList vs LinkedList

```
/*
 * ArrayLists manage arrays internally.
 * [0][1][2][3][4][5] ....
 */
List<Integer> arrayList = new ArrayList<Integer>();

/*
 * LinkedLists consists of elements where each element
 * has a reference to the previous and next element
 * [0]->[1]->[2] ....
 *    <-   <-
 */
List<Integer> linkedList = new LinkedList<Integer>();
```

ArrayList vs LinkedList



LinkedList


java.util.AbstractSequentialList<E>
java.util.LinkedList<E>
+LinkedList()
+LinkedList(c: Collection<? extends E>)
+addFirst(element: E): void
+addLast(element: E): void
+getFirst(): E
+getLast(): E
+removeFirst(): E
+removeLast(): E
Creates a default empty linked list.
Creates a linked list from an existing collection.
Adds the element to the head of this list.
Adds the element to the tail of this list.
Returns the first element from this list.
Returns the last element from this list.
Returns and removes the first element from this list.
Returns and removes the last element from this list.

ArrayList vs LinkedList

Operation \ List	LinkedList	ArrayList
get(int index)	O(n/4) average	O(1)
add(E element)	O(1)	O(1) amortized O(n) worst-case
add(int index, E element)	O(n/4) average O(1) if index is 0	O(n/2) average
remove	O(n/4) average	O(n/2) average
Iterator.remove()	O(1)	O(n/2) average
ListIterator.add(E element)	O(1)	O(n/2) average

ArrayList vs LinkedList

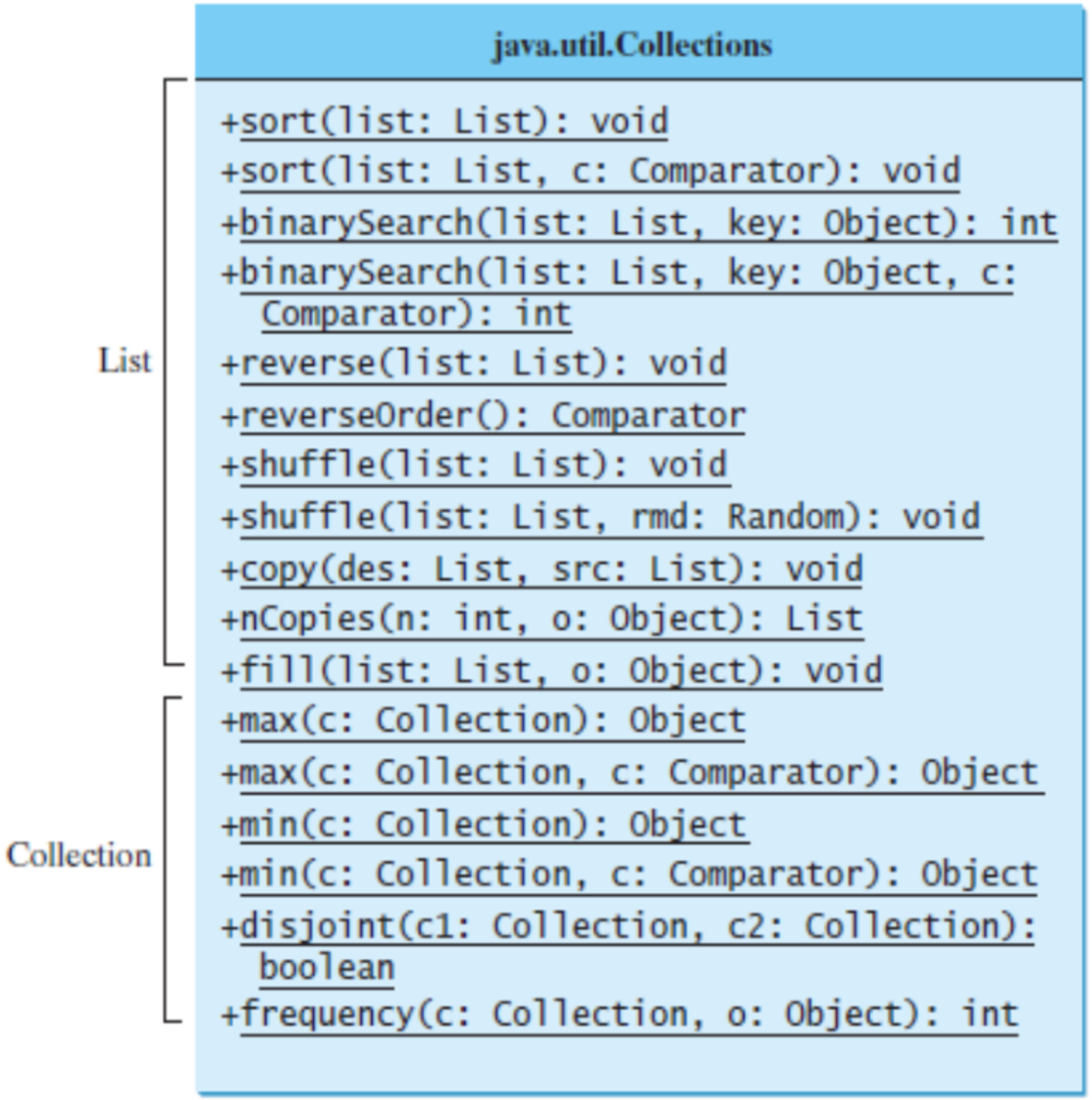
```
    doTimings("ArrayList", arrayList);
    doTimings("LinkedList" , linkedList);
}

private static void doTimings(String type, List<Integer> list) {

    for(int i=0; i<1E5; i++) {
        list.add(i);
    }

    long start = System.currentTimeMillis();

    // Add items elsewhere in list
    for(int i=0; i<1E5; i++) {
        list.add(0, i);
    }

    long end = System.currentTimeMillis();

    System.out.println("Time taken: " + (end - start) + " ms for " + type);
}
```

```
Time taken: 3296 ms for ArrayList
Time taken: 6 ms for LinkedList
```

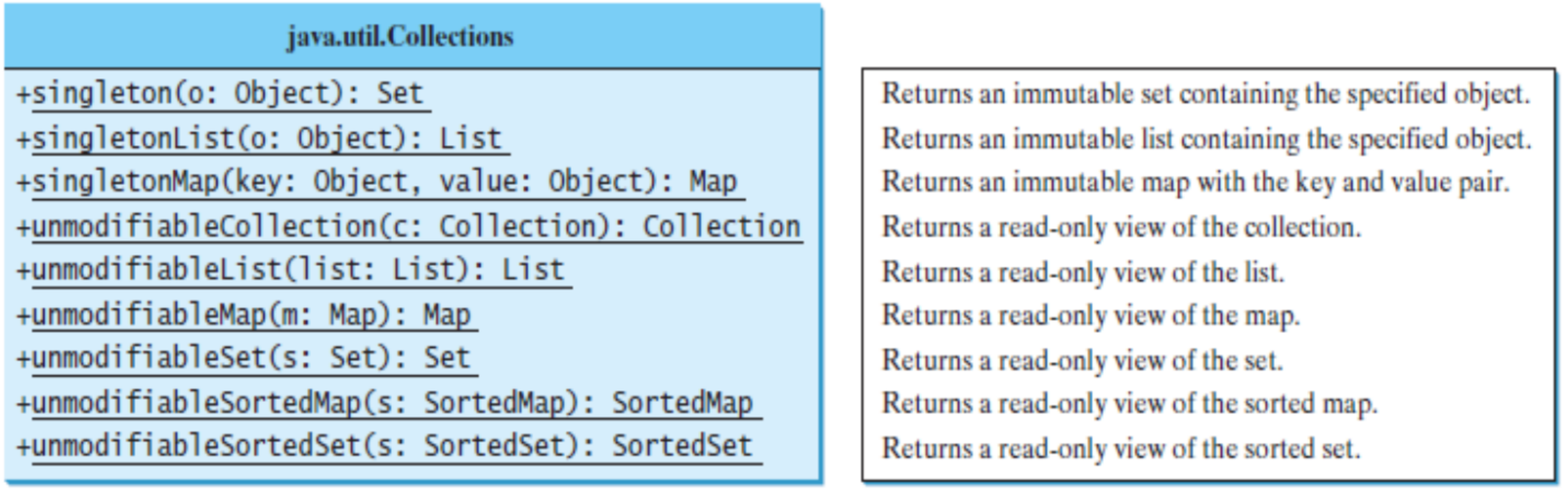
Naudingas Arrays metodas

```
List<String> list1 = Arrays.asList("red", "green", "blue");
List<Integer> list2 = Arrays.asList(10, 20, 30, 40, 50);
```

Klasė Collections, statiniai metodai



Klasė Collections, statiniai metodai



Klasė Collections, statiniai metodai

```
List<String> emptyList = Collections.emptyList();

List<Integer> singletonList = Collections.singletonList(100);

List<Integer> numbers = new ArrayList<>();
numbers.add(10);
numbers.add(12);
List<Integer> immutableList = Collections.unmodifiableList(numbers);
List<Integer> immutableListJava9 = List.of(1, 2);
```

Klasė Collections, statiniai metodai

```
// getting a mutable list
var numbers = new ArrayList<>(List.of(1, 2, 3, 2, 3, 4));

Collections.sort(numbers);    // [1, 2, 2, 3, 3, 4]
Collections.reverse(numbers); // [4, 3, 3, 2, 2, 1]
Collections.shuffle(numbers); // randomly permutes the list

//var numbers = new ArrayList<>(List.of(1, 2, 3, 2, 3, 4));
Collections.rotate(numbers, 1); // [4, 1, 2, 3, 2, 3]
Collections.rotate(numbers, 2); // [2, 3, 4, 1, 2, 3]

System.out.println(Collections.frequency(numbers, 3)); // 2
System.out.println(Collections.min(numbers)); // 1
System.out.println(Collections.max(numbers)); // 4

System.out.println(Collections.disjoint(numbers, List.of(1, 2))); // false
System.out.println(Collections.disjoint(numbers, List.of(5, 6))); // true
```

Equals() ir hashcode()

- Visi objektai įgyvendina hashCode() ir equals()
- Pagal nutylėjimą realizacijos paveldimos iš *java.lang.Object*
  - Standartinis atsako į klausimą “Ar tai tas pats objektas?” (lyg lyginant su ==)
  - Neteisingos tipiniams naudojimo atvejams
- Perrašome, tam kad du tokie patys objektai galėtų būti lygūs.
  - Paprastai įgyvendinamas lyginant laukų reikšmes.
  - IDE paprastai moka sugeneruoti

equals()

- We can override the equals method in the Circle class to compare whether two circles are equal based on their radius

```
@Override
public boolean equals(Object o) {
    if (o instanceof Circle)
        return radius == ((Circle) o).radius;
    else
        return false;
}
```

Equals()

```
public class Person {

    private String fullname;
    private int age;

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Person other = (Person) obj;
        if (age != other.age)
            return false;
        if (fullname == null) {
            if (other.fullname != null)
                return false;
        } else if (!fullname.equals(other.fullname))
            return false;
        return true;
    }
```

Equals()

```
public class Person {

    private String fullname;
    private int age;

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Person other = (Person) obj;
        return age == other.age && Objects.equals(fullname, other.fullname);
    }
}
```

Equals()

```
class BankAccount {
    String acctNumber;
    int acctType;
    public boolean equals(Object anObject) {
        if (anObject instanceof BankAccount) {
            BankAccount b = (BankAccount)anObject;
            return (acctNumber.equals(b.acctNumber) &&
                acctType == b.acctType);
        }
        else
            return false;
    }
}
```

Check whether you're comparing the same type of objects

Two bank objects are considered equal if they have the same values, for instance variables acctNumber and acctType.

hashcode()

- Panaudojamas *HashMap* objektų dėliojimui į “kibirus”
- Nuo teisingos ir geros realizacijos priklauso ar objektą bus galima teisingai panaudoti kolekcijose kurios naudoja hash kodus (HashMap, HashSet ar HashTable)
- Pasak *equals()* lygių objektų *hashCode()* turi grąžinti vienodą reikšmę (tačiau turintys vienodą hashCode, nėra garantijos jog yra lygūs).
- IDE moka sugeneruoti teisingą

hashcode()

```
private String firstName;
private String lastName;
private int age;
// constructor, getters and setters
// overridden equals method

@Override
public int hashCode() {
    int result = 17;
    result = 31 * result + (firstName == null ? 0 : firstName.hashCode());
    result = 31 * result + (lastName == null ? 0 : lastName.hashCode());
    result = 31 * result + age;
    return result;
}
// since Java 7, we have an java.util.Objects.hash(Object... values)
// utility method for hashing
```

hashcode()

```
private String firstName;
private String lastName;
private int age;

@Override
public int hashCode() {
    return Objects.hash(age, firstName, lastName);
}
```

Palyginami objektai (Comparable)

```
public static void main(String[] args) {
    List<Integer> list = new ArrayList<>();
    list.add(55);
    list.add(13);
    list.add(47);

    Collections.sort(list);
    System.out.println(list);
    //[13, 47, 55]
}
```

Palyginami objektai (Comparable)

```
public class Car {
    private int number;
    private String model;
    private String color;
    private int weight;

//constructor
//getters, setters
}
```

Palyginami objektai (Comparable)

```
List<Car> cars = new ArrayList<>();

cars.add(new Car(876, "BMW", "white", 1400));
cars.add(new Car(345, "Mercedes", "black", 2000));
cars.add(new Car(470, "Volvo", "blue", 1800));

Collections.sort(cars);
System.out.println(cars);
// The method sort(List<T>) in the type Collections
// is not applicable for the arguments (ArrayList<Car>)
```

Palyginami objektai (Comparable)

```
public class Car implements Comparable<Car>{
    private int number;
    private String model;
    private String color;
    private int weight;

    // constructor
    // getters, setters

    @Override
    public int compareTo(Car otherCar) {
        return Integer.valueOf(getNumber()).compareTo(otherCar.getNumber());
    }
    //..
```

Palyginami objektai (Comparable)

- Palyginami objektai gali būti “didesni” ar “mažesni” vieni už kitus (arba “pirmesni” ir “paskesni”)
- Palyginami objektai įgyvendina *java.util.Comparable* interfeisą
- Tokie objektai talpinami į sąrašus gali būti išrūšiuojami
- *Java.util.Comparable.* Realizuojame vieną metodą: *compareTo(Object other)*
- Jei objektai lygūs – grąžiname 0, jei esamas objektas didesnis už kitą – teigiamą skaičių (pvz. 1), jei mažesnis – neigiamą (pvz. -1)

Palyginami objektai (Comparable)

```
public class ComparablePerson implements Comparable<ComparablePerson> {

    private String fullname;
    private Integer age;
   //..
    @Override
    public int compareTo(ComparablePerson that) {
        if (this == that || this.age == that.age) {
            return 0;
        } else if (this.age > that.age) {
            return 1;
        } else if (this.age < that.age) {
            return -1;
        } else {
            throw new RuntimeException("Nesusipratimas:(");
        }
    }
```

Sąsaja Comparator<T> (1)

- Palyginimo sąsaja;
- Aprašo du metodus su dviem argumentais:
  - *compare;*
  - *equals;*
- Komparatorius naudojamas kai norime palyginti objektus, kurie neįgyvendina *Comparable* sąsajos arba norime lyginti pagal kitus kriterijus

Sąsaja Comparator<T> | Rūšiavimas (2)

```
class StringLengthComparator implements Comparator<String> {

    @Override
    public int compare(String s1, String s2) {

        int len1 = s1.length();
        int len2 = s2.length();

        if(len1 > len2) {
            return 1;
        }
        else if(len1 < len2) {
            return -1;
        }

        return 0;
    }
}
```


Sąsaja Comparator<T> | Rūšiavimas (3)

```
public class StringLengthComparator implements Comparator<String> {
    @Override
    public int compare(String s1, String s2) {
        return s1.length() - s2.length();
    }
}
```

Sąsaja Comparator<T> | Rūšiavimas (4)

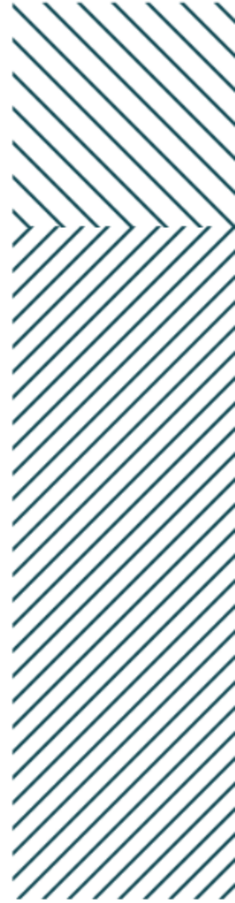
```
List<String> animals = new ArrayList<String>();

animals.add("tiger");
animals.add("lion");
animals.add("cat");
animals.add("snake");
animals.add("mongoose");
animals.add("elephant");

Collections.sort(animals, new StringLengthComparator());

for(String animal: animals) {
    System.out.println(animal);
}
```

```
cat
lion
tiger
snake
mongoose
elephant
```

Sąsaja Comparator<T> | Rūšiavimas (5)

```
List<Person> people = new ArrayList<Person>();

// Class Person: ID and name
people.add(new Person(1, "Joe"));
people.add(new Person(3, "Bob"));
people.add(new Person(4, "Clare"));
people.add(new Person(2, "Sue"));

// Sort in order of ID
Collections.sort(people, new Comparator<Person>() {
    public int compare(Person p1, Person p2) {

        if (p1.getId() > p2.getId()) {
            return 1;
        } else if (p1.getId() < p2.getId()) {
            return -1;
        }

        return 0;
    }
});
```

```
1: Joe
2: Sue
3: Bob
4: Clare
```



Sąsaja Comparator<T> | Rūšiavimas (6)

```
// Sort in order of name
Collections.sort(people, new Comparator<Person>() {
    public int compare(Person p1, Person p2) {
        return p1.getName().compareTo(p2.getName());
    }
});

for(Person person: people) {
    System.out.println(person);
}
```

```
3: Bob
4: Clare
1: Joe
2: Sue
```

Sąsaja Comparator<T> | Rūšiavimas (7)

```
@Override
public int compare(Person p1, Person p2) {
    if (p1.getName().compareTo(p2.getName()) == 0) {
        return p1.getAge() - p2.getAge();
    } else {
        return p1.getName().compareTo(p2.getName());
    }
}
```

Sąsaja Comparator<T> | Rūšiavimas (8)

```java
public class Message {

    private final String from;
    private final String content;
    private final LocalDate created;
    private int likes;

    public Message(String from, String content, int likes, String created) {
        this.from = from;
        this.content = content;
        this.likes = likes;
        this.created = LocalDate.parse(created);
    }

    // getters, setters, toString
    //...
```

Sąsaja Comparator<T> | Rūšiavimas (9)

```
public class MessageContentComparator implements Comparator<Message> {

    @Override
    public int compare(Message message1, Message message2) {
        int firstLength = message1.getContent().length();
        int secondLength = message2.getContent().length();
        return Integer.compare(firstLength, secondLength);
    }
}
```

Sąsaja Comparator<T> | Rūšiavimas (10)

```
public class MessageDateComparator implements Comparator<Message> {

    @Override
    public int compare(Message message1, Message message2) {
        return message1.getCreated().compareTo(message2.getCreated());
    }
}
```

```
public class MessageAuthorComparator implements Comparator<Message> {

    @Override
    public int compare(Message message1, Message message2) {
        return message1.getFrom().compareTo(message2.getFrom());
    }
}
```

Sąsaja Comparator<T> | Rūšiavimas (11)

```
messages.sort(new MessageContentComparator());
messages.forEach(System.out::println);
System.out.println("----------------------");
messages.sort(new MessageDateComparator());
messages.forEach(System.out::println);
```

```
2033-02-17 User76 wrote: Bump! (1)
2034-03-25 Alien wrote: Hello humans! (32)
2034-01-05 Pirate wrote: All hands on deck! (-2)
----------------------
2033-02-17 User76 wrote: Bump! (1)
2034-01-05 Pirate wrote: All hands on deck! (-2)
2034-03-25 Alien wrote: Hello humans! (32)
```

Java 8 features

```
//Vėliau rašysime tokius komparatorius:

messages.sort((m1, m2) -> m1.getCreated().compareTo(m2.getCreated()));

messages.sort(Comparator.comparing(Message::getCreated).reversed());

messages.sort(Comparator.comparing(Message::getLikes)
                        .reversed()
                        .thenComparing(Message::getFrom));
```

Java 8 features

```
//Vėliau rašysime tokius komparatorius:

messages.sort((m1, m2) -> m1.getCreated().compareTo(m2.getCreated()));

messages.sort(Comparator.comparing(Message::getCreated).reversed());

messages.sort(Comparator.comparing(Message::getLikes)
                        .reversed()
                        .thenComparing(Message::getFrom));
```