Kolekcijos

Pagrindinės paskaitos temos:

- ✓ Įvedimo-išvedimo srautai
- ✓ Objektų saugojimas (***serialization***)
- ✓ Kolekcijos (***List, Set*** interfeisai ir jų realizacijos klasės)

Įvedimo/išvedimo srautai

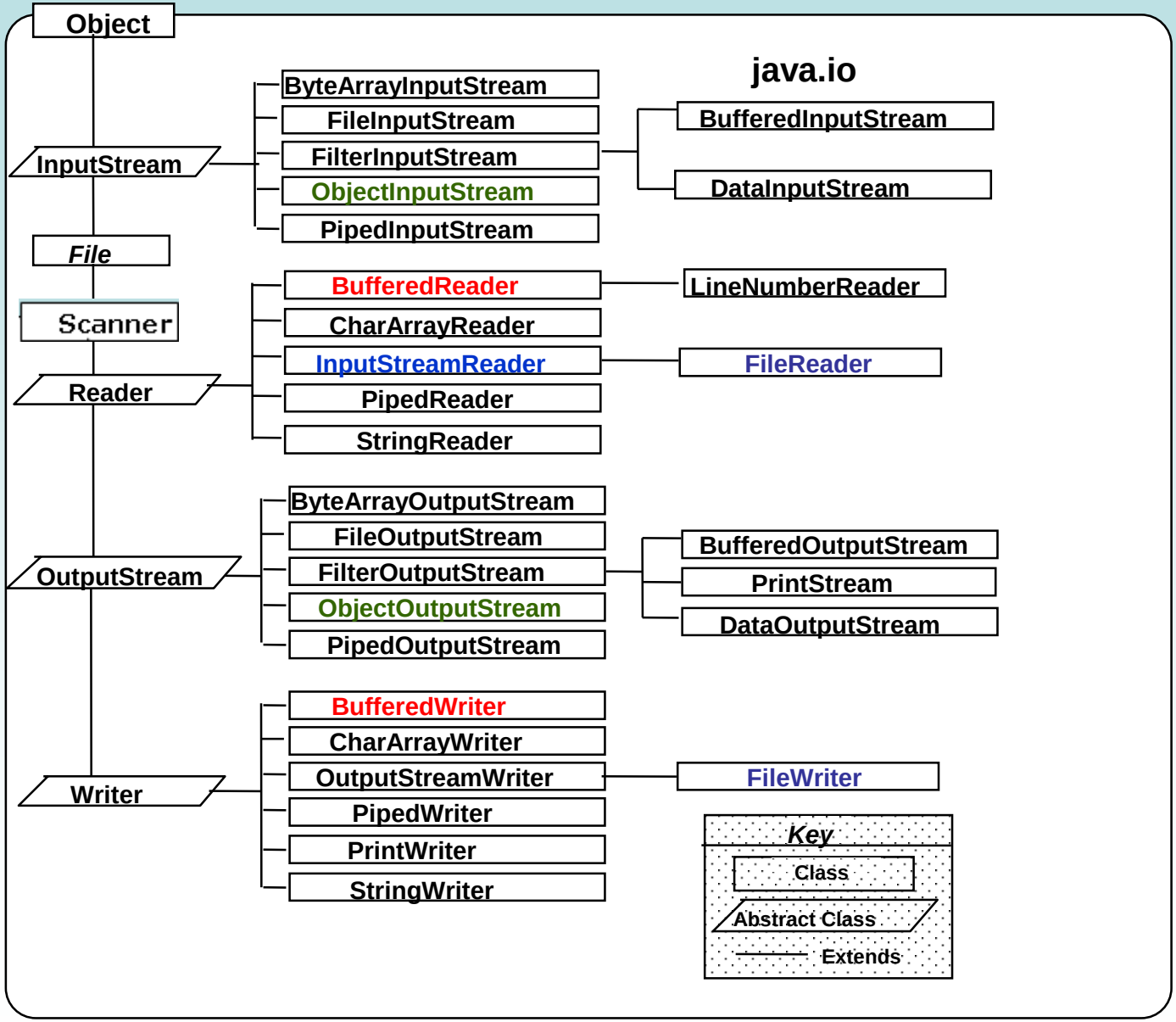
Įvedimas/išvedimas (IO) Javoje organizuotas per srautus (stream).

Standartiniai srautai operuoja su baitais.

Dirbant su Unicode simboliais (simbolis čia užima 2 baitus), patogiau dirbti su simboliniais srautais **reader** ir **writer.**

Srautas Javos kalboje atitinka klasę.

Paketo **java.io** klasės



Įvedimas metodu readline()

// Įvedimas klaviatūra:

```
BufferedReader ds = new BufferedReader(
                                    new InputStreamReader(System.in));
String s = ds.readLine();
```

// skaitymas iš tekstinio failo:

```
BufferedReader  ds = new BufferedReader(
                                    new FileReader("D:/Java/IO/Duom.dat"));
String s = ds.readLine();
```

arba:

```
File f1 = new File("D:/Java/IO/Duom.dat");          // konkreti vieta
File f1 = new File(". ", "Duom.dat");               // projekto katalogas
BufferedReader  ds = new BufferedReader(new FileReader(f1));
```

Įvedimas klasės Scanner metodais

```
import java.util.*;
-----------------------------------------------------
```

// Įvedimas klaviatūra:

```
Scanner ds = new Scanner(System.in);

String e        = ds.nextLine();        // visa eilutė bus String tipo !
String s        = ds.next();            // rezultatas bus String tipo
int k           = ds.nextInt();         // rezultatas bus int tipo
double x        = ds.nextDouble();      // rezultatas bus double tipo
```

ir kiti metodai.

// Teksto analizė:

```
String eilutė = “Honda Civic 2009 55000 88888”;
Scanner ds = new Scanner(eilutė );
```

Toliau taikomi tie patys Scanner metodai kaip ir “***Įvedimas klaviatūra”*** atveju.

* Jei duomenys neatitinka tipo, sukelia išimtį **InputMismatchException**

Skaitymas iš failo klasės Scanner metodais

```
import java.util.*;
-----------------------------------------------------
// skaitymas iš tekstinio failo:

Scanner ds = new Scanner(new File("D:/Java/IO/Duom.dat“));

Arba :

File f1 = new File("D:/Java/IO/Duom.dat");          // konkreti vieta
File f1 = new File("." , "Duom.dat");               // projekto katalogas
Scanner ds = new Scanner(f1);
```

Toliau taikomi tie patys Scanner metodai kaip ir “***Įvedimas klaviatūra”*** atveju.

* Jei duomenys neatitinka tipo, sukelia išimtį **InputMismatchException**

Rezultatų failas – metodai ir println()/print()

String s = “Honda Civic 2009 55000 88888”;

// rašymas į failą:

PrintWriter **rs** = new PrintWriter(new **FileWriter**(“D:/Java/IO/Rez.dat”));

rs.println(s);

arba:

File **f1** = new File ("D:/Java/IO/Duom.dat");
PrintWriter rs = new PrintWriter(new **FileWriter**(f1));
rs.println(s);

SVARBU importuoti ***java.io*** paketo klases:

import java.io.*;

Rašymas ištrinant arba pratęsiant toliau

FileWriter konstruktoriai failų sukūrimo metu paprastai ištrina jau esamus failus.

Tai pasiekiama naudojant standartinį konstruktorių

```
FileWriter(String failoVardas)
FileWriter(File failas)
```

Tačiau yra galimybė pridėti simbolius į esamus failus, kviečiant konstruktorių

```
FileWriter(String name, boolean papildyti)
```

Tada antro parametro reikšme nustatoma į ***true.***

```
File f1 = new File(“.” , “Duom.dat");
PrintWriter rs = new PrintWriter(new FileWriter(f1), true);
```

Papildomos klasės

File klasė turi keletą metodų darbui su konkrečiu failu (gauti failo vardą, pervardyti failą, išmesti ir t. t).

RandomAccessFile klasė leidžia tiesioginę prieigą prie failo - skaito ir rašo į tą patį failą.

StreamTokenizer klasė leidžia įvedamo simbolių srautą suskaidyti į atskirus fragmentus. Patartina naudoti konstruktorių su *Reader* tipo parametru.

StringTokenizer klasė įvedamų simbolių eilutę (***String*** tipo) skaido į atskirus fragmentus (*tokens*) pagal pasirinktus skyriklius.
Su ***StringTokenizer*** patogu analizuoti tekstą.

Standartinio konstruktoriaus skyriklių seka yra **" \t\n\r\f"** (pirmas sekoje yra tarpo simbolis), tačiau ją galima pasikeisti.

P.S: Tekstą skaidyti dar galima ir **String** klasės metodu **split** bei **Scanner** klasės metodais **next**(), **nextInt**(), nextDouble(), ...

Paketas **java.nio**

Paketas **nio** (*new io*) atsirado nuo 1.4 javos versijos. Keletas komentarų apie **nio**:

* jei standartiniame **io** pakete dirbama su baitų srautais ir simbolių srautais, tai **nio** pakete dirbama su kanalais (channels) ir buferiais (buffers). Duomenys visada skaitomi iš kanalų į buferius, o rašoma iš buferių į kanalus.

* **io** srautai dirba blokavimo rėžime, t.y. kol vykdomas *read/write* metodas, gija yra blokuojama.

* **nio** turi ir neblokuotą rėžimą, t.y., jei gija skaito duomenis iš kanalo, ji lygiagrečiai gali vykdyti dar ir kitas soperacijas.
  ** *FileChannel* kanalo tipas yra blokuotas, kiti, kaip *SocketChannel*, *ServerSocketChannel*, ne.

* **nio** leidžia viena gija valdyti keletą kanalų. Tai patogu tuo atveju, kai duomenų srautai kanaluose yra nedideli (pavyzdžiui, pokalbių serveris).

Teksto įvedimo pavyzdys - paketas **java.io**

```
import java.io.*;

public void ioIvedimas() {
        BufferedReader br = null;
        String sCurrentLine = null;
        try {
            br = new BufferedReader( new FileReader("test.txt")); // teksto failas
            while ((sCurrentLine = br.readLine()) != null) {
                System.out.println(sCurrentLine);
            }
            if (br != null) br.close();
         } catch(FileNotFoundException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
}
```

Teksto įvedimo pavyzdys - paketas **java.nio**

```
import java.io.*;
import java.nio.*;
import java.nio.channels.FileChannel;

public void nioIvedimas() {
        try {
            FileInputStream  f = new FileInputStream ("test.txt");   // teksto failas
            FileChannel inChannel = f.getChannel();
            long fileSize = inChannel.size();
            ByteBuffer buffer = ByteBuffer.allocate((int) fileSize);
            inChannel.read(buffer);
            for (int i = 0; i < fileSize; i++) {
                 System.out.print((char) buffer.get(i));
            }
            inChannel.close();
            f.close();
        } catch(FileNotFoundException e) {
            e.printStackTrace();
        } catch(IOException e) {
            e.printStackTrace();
        }
  }
```

Teksto įvedimo pavyzdys – **readAllLines** metodas

```
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.*;   // Files ir Paths klasėms
import java.util.*;

public void įvedimasFilesKlase() {
    try {
        List<String> lines = Files.readAllLines(Paths.get("test.txt"),
                                                StandardCharsets.UTF_8);
        for(String line:lines) {
                System.out.println(line);
        }

        System.out.println("Eilučių kiekis = " + lines.size());

      } catch (IOException ex) {
         System.out.println("!!! Klaida skaitant failą");
      }
    }
}
```

Objektų saugojimas (*serialization*)

Įrašius į failą simbolį 5, mes nežinome kokiu pavidalu jis ten atkeliavo – *int*, *double*, *Integer*, *String* ar dar kokiu kitu tipu. Tai todėl, kad faile turime tik duomenis, bet duomenų tipo neturime. Tai ypač nepatogu rašant / **skaitant** objekto kintamuosius.

Šiai problemai išspręsti Java turi **"objektų serializaciją"**, kai objektas atvaizduojamas baitų seka, kur saugoma:
- objekto duomenys;
- objekto tipas;
- objekto kintamųjų tipai.

Taip įrašytas į failą objektas gali būti vėliau nuskaitytas ir atkurtas atmintyje. Tam skirtos klasės **ObjectOutputStream** ir **ObjectInputStream** (perduodant konstruktoriams atitinkamai *FileOutputStream* ir *FileInputStream* objektus).

Objektų saugojimas – write/read metodai

ObjectOutput interfeisas turi metodą writeObject,
ObjectInput interfeisas turi metodą readObject :

public final **void** **writeObject**(Object obj) throws IOException

public final **Object** **readObject**() throws IOException, ClassNotFoundException

Nuskaičius objektą iš failo, **Object** tipas pakeičiamas į reikiamą objekto tipą.

Serializable interfeisas

Klasė, kurios objektai bus serializuojami, **privalo** įdiegti **Serializable** interfeisą. Šis interfeisas neturi *nei konstančių, nei metodų*, tačiau tik jo įdiegimas leidžia serializaciją (leidžia metodą *writeObject*).
Kitaip bus sukelta **NotSerializableException** išimtis.

Jei kurio nors objekto kintamojo nenorime serializuoti, tai jį deklaruojame su atributu **transient**. Pavyzdžiui, objekto kintamasis

```
private transient String pinKodas;
```

nebus įrašytas į serializacijos failą.
Nuskaičius tokį failą, šis objekto kintamasis gaus nulinę reikšmę *null* (pagal kintamojo tipą).

Pavyzdys. Paskaitoje "*DS03_Klases ir objektai.ppt*" naudojome klasę **Klientas.** Sukurkime kelis šios klasės objektus ir atlikime jų serializaciją.
Po to nuskaitykime šiuos objektus iš serializacijos failo.

Pavyzdys – klasė **Klientas**

```
import java.io.Serializable;

public class Klientas implements Serializable {
	public static final String bankoKodas="LT09"; // klientai to paties banko
	private String kodas;
	private int amžius;
	private double indėlis;

	public Klientas(String kodas, int amžius, double indėlis) {
		this.kodas = kodas;
		this.amžius = amžius;
		this.indėlis = indėlis;
	}
	@Override
	public String toString() {
		return String.format("%5s %7s %3d %9.2f", bankoKodas,
	kodas, amžius, indėlis);
    }
}
```

Pavyzdys – serializacijos klasė

```
import java.io.*;
public class RašoObjektus {
        private ObjectOutputStream out;

        public void openFile() {
                try {
                        out = new ObjectOutputStream(
                                        new FileOutputStream( "klientai.ser" ));
                } catch (IOException e) {
                        System.err.println("Klaida paruošiant failą");
                }
        }

        public void closeFile() {
                try {
                        if (out != null ) {
                                out.close();
                        }
                } catch ( IOException e ) {
                                System.err.println( "Klaida uždarant failą." );
                                System.exit( 1 );
                }
        }
```

Pavyzdys – serializacijos klasės pabaiga

```
    public void outObjects() {
        Klientas klientai[ ] = {new Klientas("SEB111", 50, 1000),
                                new Klientas("SEB476", 42, 533.20),
                                new Klientas("SWE293", 12,  23.10)};
        try {
            for (Klientas kl : klientai ) {
                System.out.println(kl); // testavimui - išveda prieš įrašymą
            }
            out.writeObject(klientai);
        }
        catch (NotSerializableException e) {
                System.err.println("Neįdiegtas Serializable interfeisas");
        } catch (IOException e) {
                System.err.println("Kita serializacijos klaida");
        }
    }

} // Klasės RašoObjektus pabaiga
```

Pavyzdys – skaitymo klasė

```
import java.io.*;
public class SkaitoObjektus {
	private ObjectInputStream read;

	public void openFile() {
		try {
			read = new ObjectInputStream(
					new FileInputStream( "klientai.ser" ));
		} catch (IOException e) {
			System.err.println("Klaida paruošiant failą");
		}
	}

	public void closeFile()		{
		try {
			if ( read != null ) {
				read.close();
			}
		} catch ( IOException e ) {
			System.err.println( "Klaida uždarant failą." );
			System.exit( 1 );
		}
	}
```

Pavyzdys – skaitymo klasės pabaiga

```
    public void loadObjects() {
        Klientas klientai[ ];
        try {
            klientai = (Klientas[ ]) read.readObject();

             for (Klientas kl : klientai ) {
                System.out.println(kl); // testas ką nuskaitė
            }
        }
        catch (EOFException e ){
                return;
        } catch (ClassNotFoundException e) {
                System.err.println("Objektas nenuskaitytas");
        } catch (IOException e) {
                System.err.println("Klaida skaitant failą");
        }
    }
} // Klasės SkaitoObjektus pabaiga
```

Galima metodą **writeObject** kviesti ir atskirai kiekvienam objektui (ne masyvui).

Jei objektai buvo surašyti ne masyvu, bet po vieną, tai galima skaityti atskirus objektus cikle iki failo pabaigos, aptinkant išimtį **EOFException**:

Pavyzdžio testas

```
public class Testas {
	public static void main(String[] args) {
		// Rašo i failą:
		System.out.println("Serializuojami klientai:");
		RasoObjektus r = new RašoObjektus();
		r.openFile();
		r.outObjects();
		r.closeFile();

		// Skaito iš serializacijos failo:
		System.out.println("\nNuskaityti klientai:");
		SkaitoObjektus s = new SkaitoObjektus();
		s.openFile();
		s.loadObjects();
		s.closeFile();

	}
}
```

Serializuojami klientai:
LT09 SEB111 50 1000,00
LT09 SEB476 42 533,20
LT09 SWE293 12 23,10

Nuskaityti klientai:
LT09 SEB111 50 1000,00
LT09 SEB476 42 533,20
LT09 SWE293 12 23,10

Java Collection Framework

Apie masyvą

Bazinė struktūra - masyvas.

Privalumai:

- didelė elementų išrinkimo sparta;
- tinka saugoti tiek objektams, tiek ir paprastiems tipams;
- sukuriant masyvą jo elementams automatiškai priskiriamos nulinės reikšmės (pagal tipą).
- galima naudoti **Arrays** klasės metodus (*fill, sort, equals, asList* ir daug kitų).

Minusai:

- fiksuotas ilgis;
- nežinoma, kiek šiuo metu užpildytas masyvas (***length*** grąžina **deklaruoto** masyvo ilgį !!!);
- visi masyvo elementai vienodo tipo (gal ir ne minusas).

Kolekcija

Kolekcija (konteineris) – tai duomenų saugykla (panaudojimo prasme).

Kolekcija – tai atitinkamą kolekcijos **interfeisą** (sąsają) įdiegiančios klasės objektas (programavimo prasme).

Kolekcija – tai "guminis" masyvas: gali "augti" tiek, kiek reikia (apimtis neribojama); gali mažėti iki tuščios.

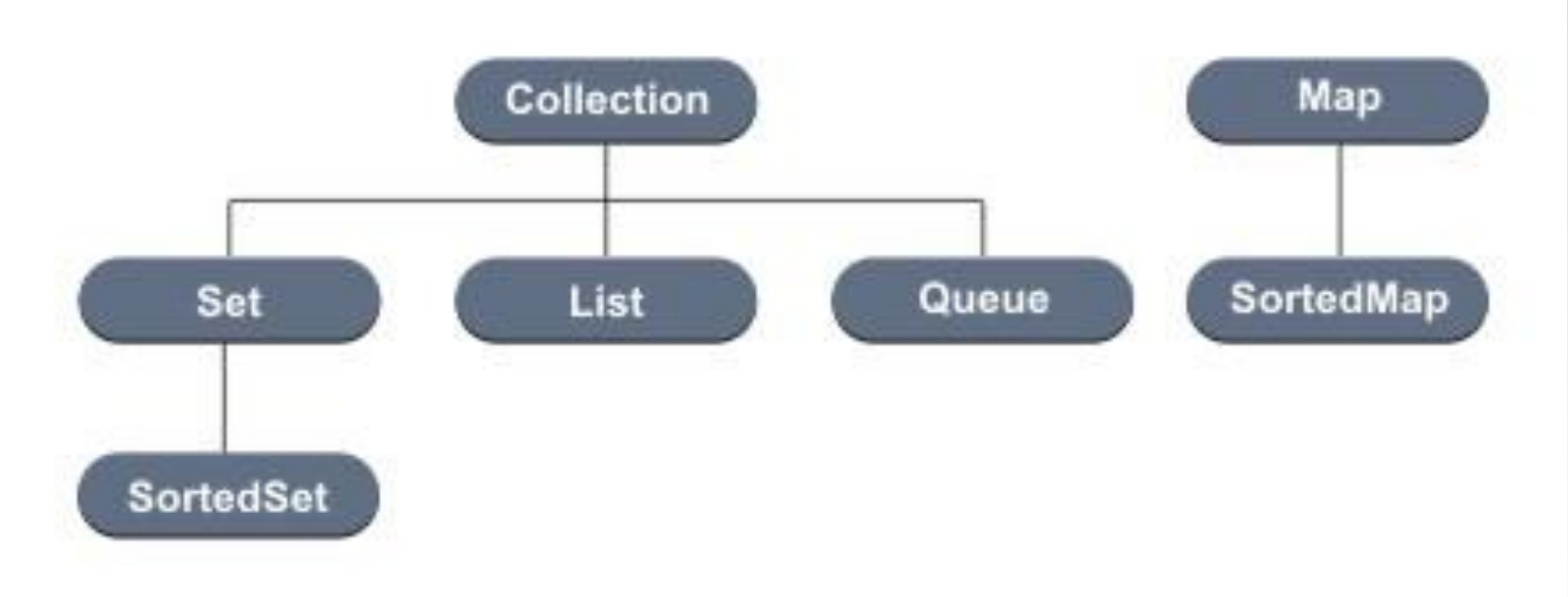
Kolekcija gali talpinti įvairių tipų objektus (**tik objektus!**).

Tai pagal ideologiją panašu *C*++ naudojamą *Standard Template Library* (STL)

Kolekcijų medis

Java turi šiuos pagrindinius kolekcijų tipus (interfeisus/sąsajas):

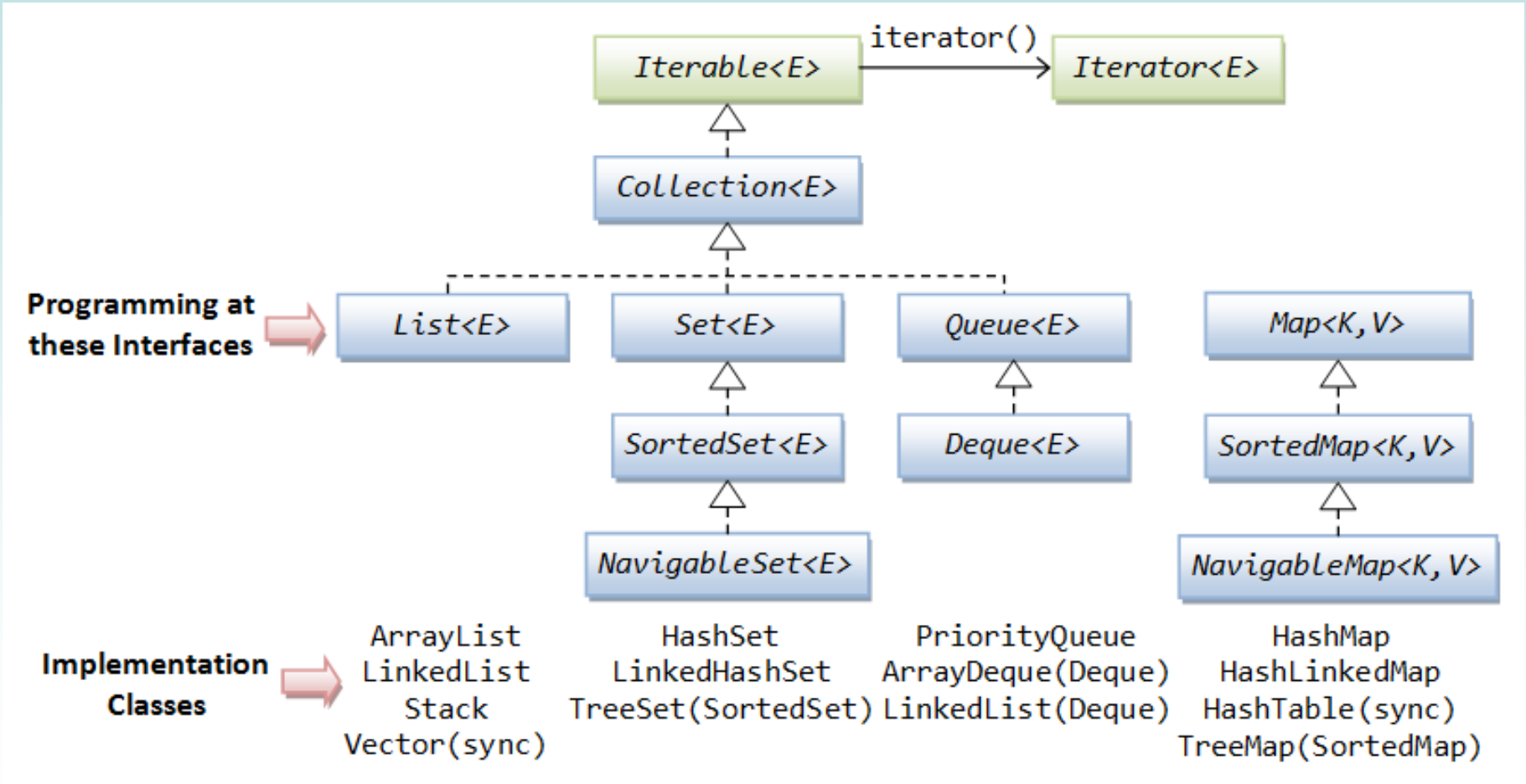
List, Set, Queue ir **Map**



* Klasės, tiesiogiai įdiegiančios ***Collection*** interfeisą, nėra.

Papildytas kolekcijų medis

Vėlenėse versijose (nuo 1.4) atsirado nauji interfeisai ir jų realizacijos klasės. Keletas iš jų:



Collection interfeiso metodai

Method Summary

boolean	**add**(E e) Ensures that this collection contains the specified element (optional operation).
boolean	**addAll**(Collection<? extends E> c) Adds all of the elements in the specified collection to this collection (optional operation).
void	**clear**() Removes all of the elements from this collection (optional operation).
boolean	**contains**(Object o) Returns `true` if this collection contains the specified element.
boolean	**containsAll**(Collection<?> c) Returns `true` if this collection contains all of the elements in the specified collection.
boolean	**equals**(Object o) Compares the specified object with this collection for equality.
int	**hashCode**() Returns the hash code value for this collection.
boolean	**isEmpty**() Returns `true` if this collection contains no elements.

Collection interfeiso metodai (tęsinys)

Iterator<E>	**iterator**() Returns an iterator over the elements in this collection.
boolean	**remove**(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	**removeAll**(Collection<?> c) Removes all of this collection's elements that are also contained in the specified collection (optional operation).
boolean	**retainAll**(Collection<?> c) Retains only the elements in this collection that are contained in the specified collection (optional operation).
int	**size**() Returns the number of elements in this collection.
Object[]	**toArray**() Returns an array containing all of the elements in this collection.
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

List interfeisas

public interface List<E> extends Collection<E>

- ✓ Interfeisas **List** saugo objektus ta tvarka, kuria jie buvo surašyti.
- ✓ Elementai gali dubliuotis.
- ✓ Elementus galima rikiuoti.

- ✓ **Paildomos operacijos**, lyginant su ***Collection*** interfeisu:

- *List* elementą galima pasiekti indeksu:

  void **add**(int *index*, E *element*)

  E **set**(int *index*, E *element*)

  E **get**(int *index*)

  E **remove**(int *index*)

List interfeisas – metodai

- elemento paieška:

  int **indexOf**(Object *obj*)

  int **lastIndexOf**(Object *obj*)

- iteratorius ***ListIterator*** (galima ne tik peržiūrėti kolekciją, bet ir modifikuoti ją – keisti, šalinti, ...):

  ListIterator<E> listIterator()

  ListIterator<E> listIterator(int index)

- poaibis:

List<E> **subList**(int from, int to)

List interfeisas - tęsinys

Kitos pastabos:

boolean **add**(E e) įdeda elementą į galą

boolean **remove**(Object o) šalina nuo pradžios

boolean **equals**(Object o) – sąrašai lygus, jei turi tuos pačius elementus ir ta pačia tvarka. Be to, sąrašų *hash* kodai turi būti lygūs. *Hash* kodai skaičiujami taip:

```
int hashCode = 1;
Iterator<E> i = list.iterator();
while (i.hasNext()) {
    E obj = i.next();
    hashCode = 31*hashCode + (obj==null ? 0 :
                                  obj.hashCode());
}
```

ListIterator interfeisas

public interface **ListIterator**<E> extends **Iterator** <E>

Metodai:

boolean **hasNext**() – *true*, jei dar yra elementas, judant pirmyn

boolean **hasPrevious**() – *true*, jei dar yra elementas, judant atgal

E **next**() – grąžina sekanį elementą

int **nextIndex**() – grąžina indeksą elemento, kurį gražintų *next()*

E **previous**() – grąžina ankstesnį elementą

int **previousIndex**() – grąžina indeksą elemento, kurį gražintų *previous()*

void **remove**() – išmeta elementą, grąžintą su *next*() arba *previous*()

void **set**(E e) – pakeičia elementą, grąžintą su *next*() arba *previous*()

void **add**(E e) – įdeda objektą į sąrašą **prieš**/po elemto, kuris būtų sekančiame žingsnyje grąžintas su *next*()/*previous*()

List interfeisas - tęsinys

Pagrindinės interfeisą **List** realizuojančios klasės yra:

ArrayList

LinkedList

Vector

ArrayList klasė

public class **ArrayList**<E> **extends** AbstractList<E>
implements **List**<E>, RandomAccess,
Cloneable, Serializable

Naudojimas: Sukuriamas interfeisą **List** įdiegiančios klasės objektas ir naudojami interfeiso **List** (ir interfeiso *Collection*) metodai:

```
ArrayList saugykla = new ArrayList();

ArrayList<Telefonas> saugykla =
                        new ArrayList<Telefonas>();
```

ArrayList panaudojimo pavyzdys

Generuoja skaičius intervale [0, 100) ir deda į ***ArrayList*** kolekciją tol, kol skaičius ne 0.

```
import java.util.*;
public class KolekcijaList {
    public static void pildoList(List mas) {
        Random r = new Random();
        int sk = 1; // bet kuri nenulinė pradinė reikšmė
        while (sk != 0) {
            sk = r.nextInt(100);
            mas.add(new Integer(sk));
        }
        System.out.println("Sugeneruota skaičių " +
                                        mas.size());
        System.out.println("Tai skaičiai: " + mas);
    }
```

ArrayList klasės panaudojimo testas

```
class Testas {
    public static void main(String[] args) {
        List<Integer> mas = new ArrayList<Integer>();
        KolekcijaList ko = new KolekcijaList();
        ko.pildoList(mas);
    }
}
```

Rezultatu pavyzdys:

Sugeneruota skaičių 10

Tai skaičiai: [92, 32, 67, 53, 28, 85, 27, 36, 25, 0]

LinkedList klasė

public class **LinkedList**<E> **extends** AbstractList<E>
implements **List**<E>, Deque<E>,
Cloneable, Serializable

Lyginant su *ArrayList*, naudojami papildomi metodai
void **addFirst(**Object o**)**
void **addLast(**Object o **)**
Object **getFirst( )**
Object **getLast( )**
Object **removeFirst( )** – grąžina ir išmeta
Object **removeLast( )**

Tai leidžia *LinkedList* tipo objektus tiesiogiai naudoti kaip steką, eilę, deką

LinkedList kaip stekas

```
public class LinkedListStekas {
	public static void main(String[] args) {
		Deque<String> st = new LinkedList<String>();
		st.push ("Vienas");
		st.push ("Du");
		st.push ("Trys");
		System.out.println("Pradinis: " + st);
		st.pop(); // šalina viršutinį
		System.out.println("Dabartinis stekas: " + st);
		System.out.println("Viršutinis elementas: " +
                                                st.getFirst());
	}
}
```

Rezultatas:

Pradinis: [Trys, Du, Vienas]
Dabartinis stekas: [Du, Vienas]
Viršutinis elementas: Du

LinkedList stekas su sava klase

```
class ManoStekas <T> {

        private Deque<T> st = new LinkedList<T>();

        public void push(T o){
                st.addFirst(o);
        }
        public T top(){
                return st.getFirst();
        }
        public T pop(){
                return st.removeFirst();
        }
        @Override
        public String toString() {
                return st.toString();
        }
}
```

LinkedList steko testas

```
public class Testas {
	public static void main(String args[]) {
		ManoStekas<String> stekas = new
							ManoStekas<String>();
		stekas.push ("Vienas");
		stekas.push ("Du");
		stekas.push ("Trys");
		System.out.println("Pradinis: " + stekas);
		stekas.pop(); // šalina viršutinį
		System.out.println("Dabartinis stekas: " +
							stekas);
		System.out.println("Viršutinis elementas: " +
							stekas.top());
	}
}
```

Rezultatas:

Pradinis: [Trys, Du, Vienas]
Dabartinis stekas: [Du, Vienas]
Viršutinis elementas: Du

Kolekcijų rikiavimai

Kolekcijoms rikiuoti, įvairioms paieškoms atlikti skirta statinių metodų klasė **Collections** **(sort, shuffle, swap, fill, copy, reverse, rotate, binarySearch, min, max,..** ir kiti metodai**).**

Yra ir metodų konkrečiam kolekcijos tipui (**emptyList, emptySet**, **checkedMap, checkedSortedMap**).

Kad panaudoti šios klasės metodą **sort**(), kolekcijos elementų klasė turi įdiegti interfeisą ***Comparable***, užklojant vienintelį jo metodą **compareTo**() (kitaip bus ***ClassCastException*** situacija).

Klasėms, įdiegiančioms šį interfeisą (*String*, *Double*..), galima tiesiogiai taikyti **sort**() metodą.

Kolekcijų rikiavimai

Kitas būdas - pasinaudoti interfeisu ***Comparator*** ir įdiegti jo metodą **compare()**. Po to šios klasės objektas perduodamas kaip papildomas parametras metodui **sort**().

Yra du sort metodai:

1. Rikiuoja elementus **natūralia tvarka**

```
public static <T extends Comparable<? super T>>
                    void sort(List<T> list)
```

("žemutinė riba": kad rikiuoti **List<T>, T** turi įdiegti interfeisą *Comparable*<***X***>, kur **X** yra **T** tipo arba vieno iš jo tėvinių tipų)

* Objektai turi būti vienodo tipo
* Tai modifikuotas **mergesort** algoritmas, garantuojantis **n log(n)** sudėtingumą.

Kolekcijų rikiavimai

2. Rikiuoja pagal **savo komparatoriaus** realizaciją. Taip pat garantuoja **n log(n)** sudėtingumą :

```
public static <T> void sort( List<T> list,
                             Comparator<? super T> c)
```

P.S: Panaudojimas analogiškas jau nagrinėtam rikiavimui su **Arrays** klasės statiniu metodu **sort**

```
public static <T> void sort(T[ ] a, Comparator<? super T> c)
```

Set interfeisas

public interface Set<E> extends Collection<E>

- ✓ Atitinka matematikoje vartojamą terminą **aibė**.
  Elementai negali dubliuotis (*null* elementas irgi gali būti tik vienas).

- ✓ Skirtumai tarp metodų realizacijų, lyginant su *Collection* interfeisu:
  - **add**(E e) metodas deda elementą tik tada, jei tokio elemento dar nera;
  - **equals**(Object o) – aibės lygios, jei jos vienodo dydžio, jei turi tuos pačius elementus ir ta pačia tvarka.
  Be to, sąrašų *hash* kodai turi būti lygūs.

Set įdiegimas – **HashSet** klasė

Interfeisą **Set** įdiegiančios klasės:

public **class** **HashSet**<E> **extends** AbstractSet<E>
implements **Set**<E>, Cloneable, Serializable

Realizuota naudojant *Hash* lentelę.

Elementai saugomi **ne pagal sudėjimo eilę**,
bet pagal specialią vidinę saugojimo tvarką
(pagal *hash* funkciją).

HashSet klasė pavyzdys

```
public class KolekcijaSet {

    private Set<Integer> aibe = new HashSet<Integer>();

    public void pildoSet() {
       Random r = new Random();
       int sk = 1;
       System.out.println("Sugeneruoti skaičiai: ");
       while (sk != 0) {
                sk = r.nextInt(100);
                System.out.print(sk + "  "); // eilinis skaičius
                aibe.add(sk);
       }
       System.out.println("\nSugeneruota skaičių " +
                                                aibe.size());
       System.out.println("Skaičiai aibėje: " + aibe);
    }
```

HashSet klasė pavyzdys (testas)

```
public static void main(String[] args) {
        KolekcijaSet ob = new KolekcijaSet();
        ob.pildoSet();
}
```

Rezultatu pavyzdys:

Sugeneruoti skaičiai:

18 37 69 59 10 4 58 56 1 25 70 46 22 16 34 81 0

Sugeneruota skaičių 17

Skaičiai aibėje: [0, 34, 69, 1, 70, 4, 37, 10, 46, 16, 18, 81, 22, 59, 25, 58, 56]

Set įdiegimas – **TreeSet** klasė

public **class** **TreeSet**<E> **extends** AbstractSet<E>
implements NavigableSet<E>,
Cloneable, Serializable

Realizuota naudojant medžio struktūtą.

Elementai saugomi ne pagal sudėjimo eilę, bet išrikiuoti pagal *compareTo*

P.S: Interfeisas **NavigableSet** paveldi interfeisą **SortedSet**, kuris paveldi interfeisą **Set**

NavigableSet interfeiso metodai

Object	**ceiling**(Object e) Returns the least element in this set greater than or equal to the given element, or if there is no such element.
Iterator	**descendingIterator**() Returns an iterator over the elements in this set, in descending order.
NavigableSet	**descendingSet**() Returns a reverse order view of the elements contained in this set.
Object	**floor**(Object e) Returns the greatest element in this set less than or equal to the given element, or if there is no such element.
SortedSet	**headSet**(Object toElement)
NavigableSet	**headSet**(Object toElement, boolean inclusive) Returns a view of the portion of this set whose elements are less than (or equal to, if is true) .
Object	**higher**(Object e) Returns the least element in this set strictly greater than the given element, or if there is no such element.
Iterator	**iterator**() Returns an iterator over the elements in this set, in ascending order.
Object	**lower**(Object e) Returns the greatest element in this set strictly less than the given element, or if there is no such element.
Object	**pollFirst**() Retrieves and removes the first (lowest) element, or returns if this set is empty.
Object	**pollLast**() Retrieves and removes the last (highest) element, or returns if this set is empty.
NavigableSet	**subSet**(Object fromElement, boolean fromInclusive, Object toElement, boolean toInclusive) Returns a view of the portion of this set whose elements range from to .
SortedSet	**subSet**(Object fromElement, Object toElement)
SortedSet	**tailSet**(Object fromElement)
NavigableSet	**tailSet**(Object fromElement, boolean inclusive) Returns a view of the portion of this set whose elements are greater than (or equal to, if is true) .

SortedSet interfeiso metodai

Method Summary

Comparator	**comparator**() Returns the comparator used to order the elements in this set, or `null` if this set uses the java.lang.Comparable of its elements.
Object	**first**() Returns the first (lowest) element currently in this set.
SortedSet	**headSet**(Object toElement) Returns a view of the portion of this set whose elements are strictly less than `toElement`.
Object	**last**() Returns the last (highest) element currently in this set.
SortedSet	**subSet**(Object fromElement, Object toElement) Returns a view of the portion of this set whose elements range from `fromElement`, inclusive, to `toElement`, exclusive.
SortedSet	**tailSet**(Object fromElement) Returns a view of the portion of this set whose elements are greater than or equal to `fromElement`.

Methods inherited from class java.util.Set

add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Methods inherited from class java.util.Collection

add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

TreeSet klasė pavyzdys (testas)

Pakeitus klasėje **KolekcijaSet** aibės aprašo eilutę į

```
private Set<Integer> aibe = new TreeSet<Integer>();
```

gautume tokius rezultatus:

Rezultatu pavyzdys:

Sugeneruoti skaičiai:

18 57 48 71 82 50 56 49 2 44 0

Sugeneruota skaičių 11

Skaičiai aibėje: [0, 2, 18, 44, 48, 49, 50, 56, 57, 71, 82]

Map interfeisas

public interface **Map<K,V>**

- ✓ Tai **raktas-reikšmė** porų lentelė, kur elementai išrenkami pagal raktą.
- ✓ Raktai kartotis negali, reikšmės – gali.
- ✓ ***Map*** elementų saugojimo tvarka nepriklauso nuo surašymo tvarkos.
- ✓ Kadangi reikšmė gali būti bet kurio tipo objektas, taip pat ir Map'o, tai galimos ir bet kokio matmens lentelės

*** Tęsinys kitose paskaitose.

“Programavimas Java. Pirmoji pažintis”, 3-ias leidimas (ebooks.ktu.lt)

9 skyrius. Failai

9.5. Objektų saugojimas

9.7. Objektų masyvo rikiavimas

12 skyrius. Kolekcijos (objektų saugojimo būdai)

Sekančios paskaitos temos:

Map interfeisas plačiau
Duomenų struktūrų pasirinkimo metodika
Kolekcijų atminties sunaudojimas
Greitaveika