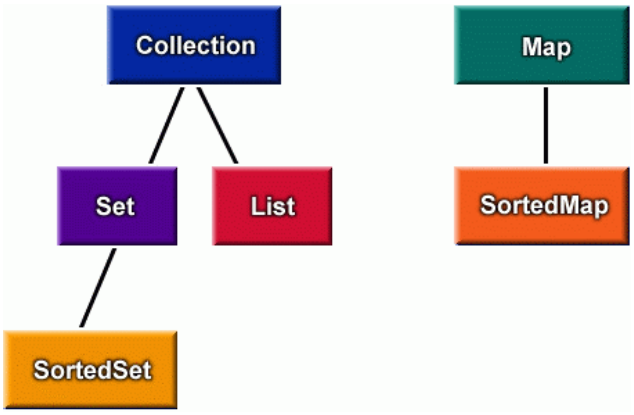
Kolekcijos (objektų saugojimo būdai)

Masyvas – tai pats paprasčiausias "indas" objektams saugoti. Masyvas javoje yra objektas.
Privalumai:
- diddelis elementų išrinkimo greitis;
- leidžia saugoti tiek objektus, tiek ir paprastus tipus;
- sukuriant elementams automatiškai priskiriamos nulinės reikšmės (pagal tipą).

Minusai:
- fiksoutas masyvo ilgis;
- nežinomas dabartinis msyvo užpildymas (length grąžina visą masyvo ilgį);
- visi masyvo objektai vienodo tipo.

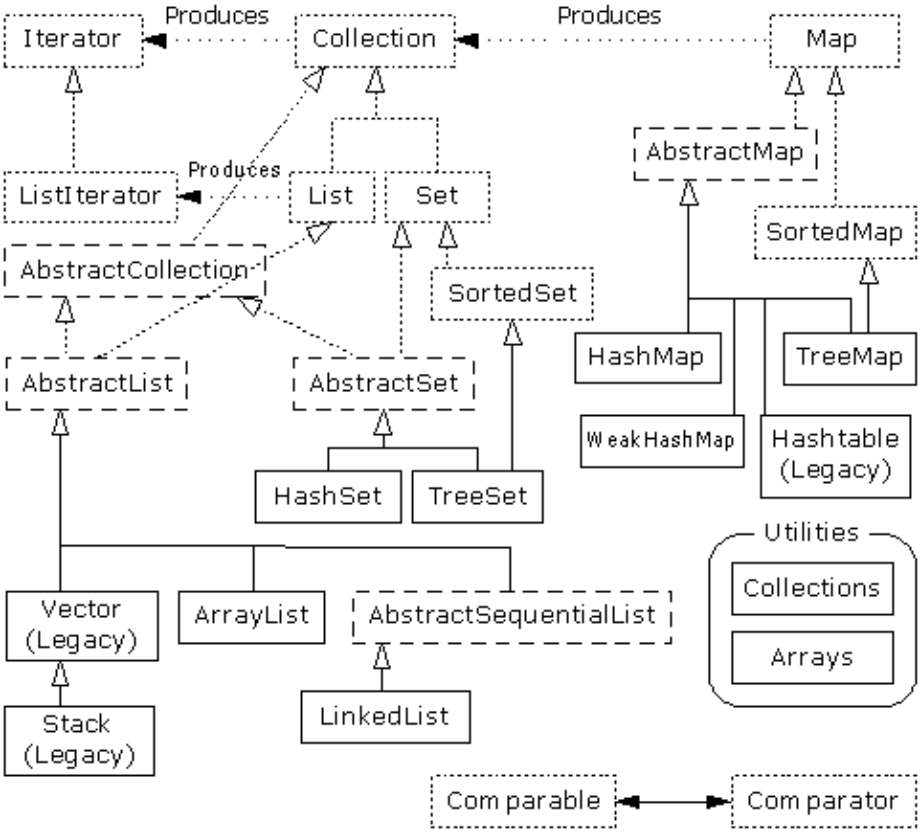
Java leidžia grąžinti msyvą (masyvą-objektą) metodo vardu ir jis gyvous tol, kol jis bus reikalingas (C++ leidžia grąžinti tik rodyklę, o tai sudaro problemų dėl grąžinto masyvo egzistavimo trukmės).

Java turi trijų tipų kolekcijas - tikliau tris interfeisus : **Set**, **List** ir **Map** (piešinukas rodo visų kolekcijų interfeisų medį) :



Pagrindinis kolekcijų minusas - patalpinus objektą, prarandama informacija apie jo tipą. Todėl kolekcijos saugo *Object* tipo rodykles. Tačiau (**cast**) operacijos dėka visada galima nuskaitytą tipą persivesrti į reikiamą (aišku tam reikia žinoti į kokį).

Pilnesnė kolekcijų interfeisų ir klasiu hierahijos lentelė:



1. public interface **List** extends *Collection*

Klasės, tiesiogiai realizuojančios *Collection* interfeisą nėra.

List saugo objektus ta tvarka, kokia jie buvo surašyti. Elementus galima tiek rikiuoti, tiek ir dubliuoti (dublis reiškia, jog ir tipas ir reikšmė sutampa). *List* elementą palima pasiekti tiek per indeksą, tiek ir per reikšmę. Padrindinės interfeisą **List** realicuojančios klasės – **ArrayList, LinkedList** ir **Vector** (pasenusi).

ArrayList – tai pats paprasčiausias automatiškai pasiilginantis masyvas, kurio panaudojimas ypatingai paprastas:

- sukuriamas **ArrayList** tipo objektas;
- elementui talpinti panaudoti metodą **add**(), paėmimui – **get**(int indeksas);
- metodas **size**() grąžina dabartinį elementų kiekį masyve.

Labai greiatas tiesioginis elemento išrinkimas (pagal indeksą), bet lėtos eklementų įterpimo ir šalinimo operacijos masyvo viduryje.

LinkedList greitas nouseklaus apdorojimo operacijose ir eklementų įterpimo bei šalinimo operacijose masyvo viduryje. Letas tiesioginio išrinkimo operacijose. Lyginant su **ArrayList,** turi papildomus metodus **addFirst( ), addLast( ), getFirst( ), getLast( ), removeFirst( ), removeLast(**), kas leidžia **LinkedList** tipo objektus tiesiogiai naudoti kaip steką, eilę ir deką.

Klasių prototipai :

public class **ArrayList** extends AbstractList implements List, Cloneable, Serializable
public class **LinkedList** extends AbstractSequentialList implements List, Cloneable, Serializable
public class **Vector** extends AbstractList implements List, Cloneable, Serializable

```
// PVZ1:
// Generouti sveikus atsitiktinius skaičius ir surašyti į neriboto
//   ilgio ArrayList tipo masyvą. Baigti, kai sugeneroutas skaičius nulis.
//   Išvesti sugeneroutas skaičius į ekraną.
import java.util.*;
class Metodai {
   private int maxReiksme;
   Metodai(int max) {
      maxReiksme = max;
   }
   private Random r = new Random();
   public int randInt() {
      // grazina reiksme is intervalo [0, maxReiksme]
      return r.nextInt(maxReiksme);
   }
}
public class GuminisMasyvas {
   public static void main(String[] args) {
      List masyvas = new ArrayList();
      Metodai met = new Metodai(100);
      int skaicius = 1; // bet kokia nenukine pradine reiksme
      while (skaicius != 0) {
         skaicius = met.randInt();
         Integer intOb = new Integer(skaicius);
         masyvas.add(intOb);
         if (masyvas.size() > 1000) {
            break; // nebutina, tiesiog siam pvz dau skaiciu nereikia
         }
      }
      System.out.println("Sugeneruota skaiciu " + masyvas.size());
      System.out.println("Tai skaiciai :");
      for (int i = 0; i < masyvas.size(); i++) {
         System.out.print(masyvas.get(i) + "  ");
      }
   }
}

// PVZ2 :
// Nuosavas stekas LinkedList klasės pagrindu :
class Stekas1 {
   private LinkedList list = new LinkedList();
   public void push(Object v) {
      // patalpina elementa
      list.addFirst(v);
   }
   public Object top() {
      // skaito steko virsutini elementa
      return list.getFirst();
   }
```

```
   public Object pop() {
      // nuskaito steko virsutini elementa ir pasalina is steko
      return list.removeFirst();
   }
}
class Metodai {
   private int maxReiksme;
   Metodai(int max) {
      maxReiksme = max;
   }
   private Random r = new Random();
   public int randInt() {
      // grazina reiksme is intervalo [0, maxReiksme]
      return r.nextInt(maxReiksme);
   }
}

public class Stekas {
   public static void main(String[] args) {
      Metodai met = new Metodai(100);
      Stekas1 stekas = new Stekas1();
      System.out.println("\nElementai, talpinami i steka:");
      for(int i = 0; i < 10; i++) {
         skaicius = met.randInt();
         Integer intOb = new Integer(skaicius);
         System.out.print(intOb + "  ");
         stekas.push(intOb);
      }
      System.out.println("\nBandymeliai :");
      System.out.println(stekas.top());
      System.out.println(stekas.top());
      System.out.println(stekas.pop());
      System.out.println(stekas.pop());
      System.out.println(stekas.pop());
   }
}
```

Interfeiso ***List*** metodai :

Method Summary	
void	**add**(int index, Object element) Inserts the specified element at the specified position in this list (optional operation).
Boolean	**add**(Object o) Appends the specified element to the end of this list (optional operation).
Boolean	**addAll**(Collection c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
Boolean	**addAll**(int index, Collection c) Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
void	**clear**()

	Removes all of the elements from this list (optional operation).
Boolean	**contains**(Object o) Returns true if this list contains the specified element.
Boolean	**containsAll**(Collection c) Returns true if this list contains all of the elements of the specified collection.
Boolean	**equals**(Object o) Compares the specified object with this list for equality.
Object	**get**(int index) Returns the element at the specified position in this list.
int	**hashCode**() Returns the hash code value for this list.
int	**indexOf**(Object o) Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.
Boolean	**isEmpty**() Returns true if this list contains no elements.
Iterator	**iterator**() Returns an iterator over the elements in this list in proper sequence.
int	**lastIndexOf**(Object o) Returns the index in this list of the last occurrence of the specified element, or -1 if this list does not contain this element.
ListIterator	**listIterator**() Returns a list iterator of the elements in this list (in proper sequence).
ListIterator	**listIterator**(int index) Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in this list.
Object	**remove**(int index) Removes the element at the specified position in this list (optional operation).
Boolean	**remove**(Object o) Removes the first occurrence in this list of the specified element (optional operation).
Boolean	**removeAll**(Collection c) Removes from this list all the elements that are contained in the specified collection (optional operation).
Boolean	**retainAll**(Collection c) Retains only the elements in this list that are contained in the specified collection (optional operation).
Object	**set**(int index, Object element) Replaces the element at the specified position in this list with the specified element (optional operation).
int	**size**() Returns the number of elements in this list.
List	**subList**(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
Object[]	**toArray**() Returns an array containing all of the elements in this list in proper sequence.
Object[]	**toArray**(Object[] a) Returns an array containing all of the elements in this list in proper sequence; the runtime type of the returned array is that of the specified array.

2. public interface **Set** extends *Collection*

Set neleidžia nei elementus rikiuoti, nei jų dubliuoti. **Set** saugo tik unikalius objektus, be to tam naudojama speciali vidine saugojimo tvarka. Interfeisą **Set** realicuojančios pagrindinės klasės yra **HashSet** it **TreeSet.**

HashSet tinka atvejui, kai reikalingas didelis paieškos greitis. Objektams būtinas **hashCode( )** metodas.

TreeSet sugo objektus medžio tipo pavidalu. Turi specialius metodus darbui su medžio šakomis (**first**(), **last**(), **headSet**(), **subSet**(), **tailSet**()).

Klasių prototipai :

public class **HashSet** extends AbstractSet implements Set, Cloneable, Serializable
public class **TreeSet** extends AbstractSet implements SortedSet, Cloneable, Serializable

Interfeiso ***Set*** metodai :

Method Summary	
Boolean	**add**(Object o) Adds the specified element to this set if it is not already present (optional operation).
Boolean	**addAll**(Collection c) Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	**clear**() Removes all of the elements from this set (optional operation).
Boolean	**contains**(Object o) Returns true if this set contains the specified element.
Boolean	**containsAll**(Collection c) Returns true if this set contains all of the elements of the specified collection.
Boolean	**equals**(Object o) Compares the specified object with this set for equality.
int	**hashCode**() Returns the hash code value for this set.
Boolean	**isEmpty**() Returns true if this set contains no elements.
Iterator	**iterator**() Returns an iterator over the elements in this set.
Boolean	**remove**(Object o) Removes the specified element from this set if it is present (optional operation).
Boolean	**removeAll**(Collection c) Removes from this set all of its elements that are contained in the specified collection (optional operation).
Boolean	**retainAll**(Collection c) Retains only the elements in this set that are contained in the specified collection (optional operation).
int	**size**() Returns the number of elements in this set (its cardinality).
Object[]	**toArray**() Returns an array containing all of the elements in this set.
Object[]	**toArray**(Object[] a) Returns an array containing all of the elements in this set whose runtime type is that of the specified array.

3. public interface **Map**

Map - tai **reikšmė-raktas** porų lentelė (žodynėlis), kur elementai išrenkami pagal raktą. Raktai kartotis negali, reikšmės – gali. **Map** elementų saugojimo tvarka taip pat nepriklauso nuo surašymo tvarkos. Kadangi reikšme gali buti bet kokio tipo objektas, tame tarpe ir Map'o, tai galimos ir bet kokio išmatavimo lentelės. Nauji elementai gali būti sudedami metodu **put** (Object raktas, Object reiksme), o paimami metodu **get**(Object raktas). Pagrindinės interfeisą **Map** realicuojančios klasės yra **HashMap, TreeMap** ir **Hashtable** (pasenusi).

HashMap lentelėje elemento paieška ir įterpimas yra gana greitas (šis laikas yra pastovus).

TreeMap objektus saugo specialioje medžio tipo struktūroje. Turi papildomus metodus darbui su šio medžio dalimis (**firstKey**(), **lastKey**(), **subMap**(), **tailMap**()).

Klasių prototipai :

public class **Hashtable** extends Dictionary implements Map, Cloneable, Serializable
public class **HashMap** extends AbstractMap implements Map, Cloneable, Serializable
public class **TreeMap** extends AbstractMap implements SortedMap, Cloneable, Serializable

```
// PVZ3:
// Sudaryti ir atspauzdinti lentele. Raktas – atsitiktiniai Integer tipo skaičiai. Reikšmės – atsitiktiniai
// skaičiai, pervesti į String tipo objektus.
import java.util.*;
public class Zodinelis {
   public static void main(String[] args) {
      HashMap hm = new HashMap();
      for(int i = 1; i <= 10; i++) {
         // Raktui generousime skaicius tarp 0 ir 25,
         // reiksmei - simbolius tarp 0 ir 50.
         //  (statinis Math klases metodas random() generouja intervale [0, 1]
         Integer raktas = new Integer((int)(Math.random() * 25));
         if (hm.containsKey(raktas) == false) {
            String simbolineReiksme = Integer.toString((int)(Math.random() * 50));
            hm.put(raktas, simbolineReiksme);
         }
      }
      // isveda lentele poromis  raktas="reiksme" :
      System.out.println("Tai Map lentele :");
      System.out.println(hm); // isveda poromis raktas=reiksme

   }
}
```

Raktu gali buti bet kurios klasės objektai. Tačiau kad padidinti išrinkimo greitį, raktai užkodoujami specialiu sveikaskaitiniu kodu (hash code). Tai atlieka Object klasės metodas hashCode(). Todėl kai raktu naudojamos standartines java klasės jokių problemų neiškyla. Brt jeigu sugalvosite raktu naudoti savo sukurtą klasę, teks perkrauti metodus hashCode() ir equals() .

Interfeiso ***Mapt*** metodai :

Inner Class Summary	
Static interface	**Map.Entry** A map entry (key-value pair).

Method Summary	
void	**clear**() Removes all mappings from this map (optional operation).
boolean	**containsKey**(Object key) Returns true if this map contains a mapping for the specified key.
boolean	**ContainsValue**(Object value) Returns true if this map maps one or more keys to the specified value.
Set	**entrySet**() Returns a set view of the mappings contained in this map.
boolean	**equals**(Object o) Compares the specified object with this map for equality.
Object	**get**(Object key) Returns the value to which this map maps the specified key.
int	**hashCode**() Returns the hash code value for this map.
boolean	**isEmpty**() Returns true if this map contains no key-value mappings.
Set	**keySet**() Returns a set view of the keys contained in this map.
Object	**put**(Object key, Object value) Associates the specified value with the specified key in this map (optional operation).
void	**putAll**(Map t) Copies all of the mappings from the specified map to this map (optional operation).
Object	**remove**(Object key) Removes the mapping for this key from this map if present (optional operation).
int	**size**() Returns the number of key-value mappings in this map.
Collection	**values**() Returns a collection view of the values contained in this map.

Rikiavimai ir paieškos

Kolekcijų rikiavimui, įvairioms paieškoms atliktu skirta statinių metodų klase *Collections* :

public class **Collections** extends Object (paketas util)

Tačiau kad panaudoti jos metodą **sort**(), kolekcijos elementai privalio realizuoti interfeisą *Comparable* (jis turi tik vieną metodą *copmareTo*) :

public interface **Comparable**

Kitaip gausite ***ClassCastException*** situaciją. Šį interfeisą realizuoja tokios klasės :

Byte, Character, Double, File, Float, Long, ObjectStreamField, Short, String, Integer, BigInteger, BigDecimal, Date, CollationKey

Štai keletas **Collection** klasės metodų :

Method Summary

Static int	**binarySearch**(List list, Object key) Searches the specified list for the specified object using the binary search algorithm.
Static int	**binarySearch**(List list, Object key, Comparator c) Searches the specified list for the specified object using the binary search algorithm.
static void	**copy**(List dest, List src) Copies all of the elements from one list into another.
static Enumeration	**enumeration**(Collection c) Returns an enumeration over the specified collection.
static void	**fill**(List list, Object o) Replaces all of the elements of the specified list with the specified element.
static Object	**max**(Collection coll) Returns the maximum element of the given collection, according to the *natural ordering* of its elements.
static Object	**max**(Collection coll, Comparator comp) Returns the maximum element of the given collection, according to the order induced by the specified comparator.
static Object	**min**(Collection coll) Returns the minimum element of the given collection, according to the *natural ordering* of its elements.
static Object	**min**(Collection coll, Comparator comp) Returns the minimum element of the given collection, according to the order induced by the specified comparator.
static List	**nCopies**(int n, Object o) Returns an immutable list consisting of n copies of the specified object.
static void	**reverse**(List l) Reverses the order of the elements in the specified list.
static Comparator	**reverseOrder**() Returns a comparator that imposes the reverse of the *natural ordering* on a collection of objects that implement the Comparable interface.
static void	**shuffle**(List list) Randomly permutes the specified list using a default source of randomness.
Static void	**shuffle**(List list, Random rnd) Randomly permute the specified list using the specified source of randomness.
Static Set	**singleton**(Object o) Returns an immutable set containing only the specified object.
Static void	**sort**(List list) Sorts the specified list into ascending order, according to the *natural ordering* of its elements.
Static void	**sort**(List list, Comparator c) Sorts the specified list according to the order induced by the specified comparator.

Elementai bus surikiuti naturalia eile, t.y :

Class	Natural Ordering
Byte	signed numerical
Character	unsigned numerical
Long	signed numerical
Integer	signed numerical
Short	signed numerical
Double	signed numerical
Float	signed numerical
BigInteger	signed numerical
BigDecimal	signed numerical
File	system-dependent lexicographic on pathname.

`String`	Lexicographic
`Date`	Chronological
`CollationKey`	locale-specific lexicographic

Kitas rikiavimo būdas - pasinaudoti interfeisu *Comparator* ir realizuoti jo metodą *compare().* Antro interfeiso metodo *equals()* realizuoti nebūtina, nes jis realizuotas klasėje *Object.* Po to šios implementacijos klasės objektas perduodamas kaip papildonas parametras metodui *sort().*

Surikioutame masyve (kitaip rezultatas neprognozuojamas) galima naudoti greitus paieškos metodus *Arrays.**binarySearch***().

Masyvams rikiuoti egzituoja analogiška klasė *Arrays* (paketas util):

Aišku, kai masyvai susideda is paprastu tipų (ne iš įrasų su skirtingais laukais), pilnai rikiavimui pakanka ir javos pakete esančiu metodų compare(), compareTo(), equals() realizacijų. Be to, šių metodų realizacijos yra pakankamai optimalios (paprastiems tipams naudojamas greitas rikiavimas, o objektams – specialus rikiavimas suliejimu).

```
import java.util.*;

public class EiluciuRikiavimas {
        public static void main(String[] args) {
                String[] sa = new String[100];
                // masyvo užpildymas.
                Arrays.sort(sa);
        }
}
```

Taigi, jei šis "leksikografinis" rikiavimas patenkina (kai pradžioje eina žodžiai iš didžiųjų raidžių, o po to iš mažųjų), tai viskas gerai. Norint pakeisti rikaivimo būdą, pvz., rikiuoti nevertinant raidžių registro, pakaks perdengti standartinį *compare()* metodą savu:

```
public class MonoRikiavimas implements Comparator {
        public int compare(Object o1, Object o2) {
                String s1 = (String)o1;
                String s2 = (String)o2;
                return s1.toLowerCase().compareTo(s2.toLowerCase());
        }
}
public class EiluciuRikiavimas {
        public static void main(String[] args) {
                String[] sa = new String[100];
                // masyvo užpildymas.
                Arrays.sort(sa, new MonoRikiavimas());
        }
}
```

Papildomai pažiūrėti :

(1) http://developer.java.sun.com/developer/onlineTraining/Programming/BasicJava2/collec.html

(2) http://java.sun.com/docs/books/tutorial/collections/index.html

(3) Bruce Eckel, Thinking in Java, 2nd edition, Revision 12, **9-as skyrius**