Java Collection’s

Mindaugas Karpinskas

http://beginnersbook.com/java-collections-tutorials/

http://www.javatpoint.com/collections-in-java

https://docs.oracle.com/javase/tutorial/collections/

Collection – talpykla

• Objektas – kolekcija / aibė / sąrašas..., užtikrinantis objektų saugojimą, paiešką,...

• Java Collection klasių šeima pateikia standartinių duom. struktūrų bei algoritmų realizacijas

• Operuojama objektais, bet ne primityviaisiais tipais

Vystymasis

- Javoje 1.x ribotos priemonės – Vector, Hashtable, Enumeration
- 1.2 – Collection klasės, vieningas požiūris
- 1.5 – Parametrizacija, papildomos klasės (Queue), primit. tipų – Apvalkalų automatinė konversija (Integer <-> int)

Important!!!

Object **equals**

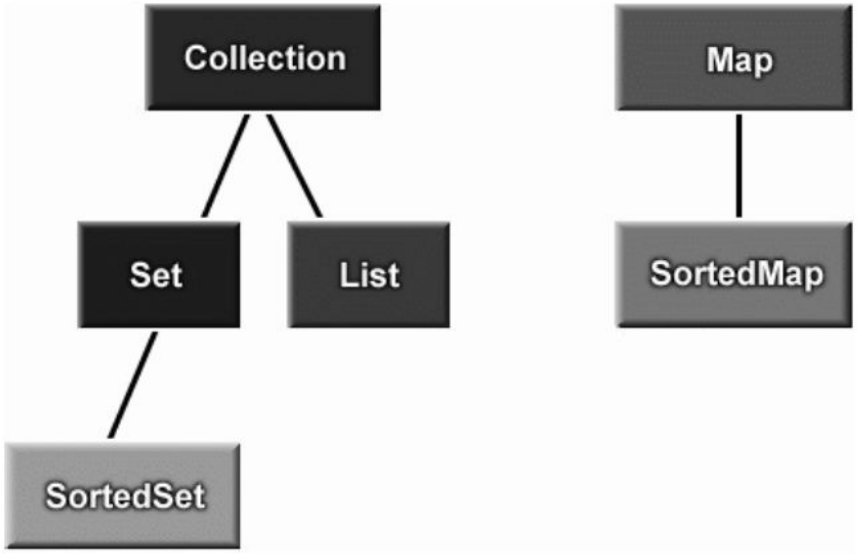
Object **hashCode**

A **hashcode** is a number generated from any object. This is what allows objects to be stored/retrieved quickly in a Hashtable.

Imagine the following simple example: On the table in front of you you have nine boxes, each marked with a number 1 to 9...

http://www.javaworld.com/article/2074996/hashcode-and-equals-method-in-java-object---a-pragmatic-concept.html

Interfeisai



– Apibrėžti **java.util** pakete

– Yra realizuojančiosios klasės

Ypatybės

- ***Collection*** – objektų elementų rinkinys
- ***Set*** – rinkinys be pasikartojančių elementų
- ***List*** – elementų seka
- ***Map*** – porų <raktas, reikšmė> aibė
- ***SortedSet***
- ***SortedMap***
- Pastaba: Objektai rikiuojami realizuojant *Comparable, Comparator* interfeisus.

Collection interfeisas

- Nusako bendriausias operacijas
- Peržiūra iteratoriumi

```
public interface Collection {
    // ------ Basic Operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(Object element);    // Optional
    boolean remove(Object element); // Optional
    Iterator iterator();
    // ------- Bulk Operations
    boolean containsAll(Collection c);
    boolean addAll(Collection c);    // Optional
    boolean removeAll(Collection c); // Optional
    boolean retainAll(Collection c); // Optional
    void clear();                    // Optional
    // ------- Array Operations
    Object[] toArray();
    Object[] toArray(Object a[]);
}
```

```
public interface Iterator {
    boolean hasNext();
    Object next();
    void remove();    // Optional
}

//================================//

static void filter(Collection c) {
    for (Iterator i = c.iterator(); i.hasNext(); )
        if (!cond(i.next()))
            i.remove();
}
```

Set interfeisas

- Neturi naujų metodų.
- Nauja semantika: pasikartojantys elementai draudžiami.

List - sąrašas

- Nuosekli elementų seka
- Kreiptis pagal indeksą
- Paieška grąžina elemento poziciją
- Išplėstas iteratorius
- Posarašiai (Range-view)
- Realizuoja ArrayList, LinkedList
- Collections klasė įgalina pritaikyti įvairius algoritmus List egzemplioriams

```
public interface List extends Collection {
   //----- Positional Access
   Object get(int index);
   Object set(int index, Object element);      // Optional
   void add(int index, Object element);        // Optional
   Object remove(int index);                   // Optional
   abstract boolean addAll(int index, Collection c); // Opt
   //----- Search
   int indexOf(Object o);
   int lastIndexOf(Object o);
   //----- Iteration
   ListIterator listIterator();
   ListIterator listIterator(int index);
   // Range-view
   List subList(int from, int to);
}
```

List’o panaudojimas

```
private static private static void swap(List a, int i, int j) {

Object tmp = a.get(i);

a.set(i, a.get(j));

a.set(j, tmp);

}
```

ArrayList

```
public static void main(String[] args) {
    List<String> kolekcija = new ArrayList<>();
    kolekcija.add("Pirmas");
    kolekcija.add("Antras");
    kolekcija.add("Pirmas");
    System.out.println(kolekcija.get(2));
}
```

U1

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

1. Pridėti 5 elemntus
2. Patestuoti
    a. get()
    b. size()
    c. remove()
    d. set(int index,obj)
    e. indexOf()

ArrayList

```
public static void main(String[] args) {
    List<String> kolekcija = new ArrayList<>();
    kolekcija.add("Pirmas");
    kolekcija.add("Antras");
    kolekcija.add("Pirmas");
    System.out.println(kolekcija.get(2));
}
```

HashSet

```
public static void main(String[] args) {
    Set<String> kolekcija = new HashSet<>();
    kolekcija.add("Pirmas");
    kolekcija.add("Antras");
    kolekcija.add("Pirmas");
    System.out.println(kolekcija.size());
}
```

U2

Set<String> kolekcija = new HashSet<String>();

1. Pridėti 5 elemntus
2. Patestuoti
   a. size()
   b. remove()
3. Pridėti dar elementų su tomis pačiomis reikšmėmis
4. Patestuoti: size() remove()

U3

Metodas, kuris išspausdintų visas argumento (Collection<String> kolekcija) reikšmes

Map

- Susieja raktą su reikšme
- Įgyvendina: HashMap, TreeMap

```
public interface Map {

    // Basic Operations
    Object put(Object key, Object value);
    Object get(Object key);
    Object remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();

    // Bulk Operations
    void putAll(Map t);
    void clear();
```

```
// Collection Views
    public Set keySet();
    public Collection values();
    public Set entrySet();

    // Interface for entrySet elements
    public interface Entry {
        Object getKey();
        Object getValue();
        Object setValue(Object value);
    }
} // End Map
```

Realizacijos

		Implementations			
		Hash Table	**Resizable Array**	**Balanced Tree**	**Linked List**
Interfaces	**Set**	HashSet		TreeSet	
	List		ArrayList		LinkedList
	Map	HashMap		TreeMap	

U4

Map<Integer, String> kolekcija = new HashMap<>();

1. Pridėti 5 elemntus : 1,”Vienas” ...
2. Patestuoti
    a. get()
    b. size()
    c. remove()
    d. put
    e. keySet
3. Pridėti dar elementų su tomis pačiomis reikšmėmis
4. Patestuoti: get() size() remove() put()

Iterator

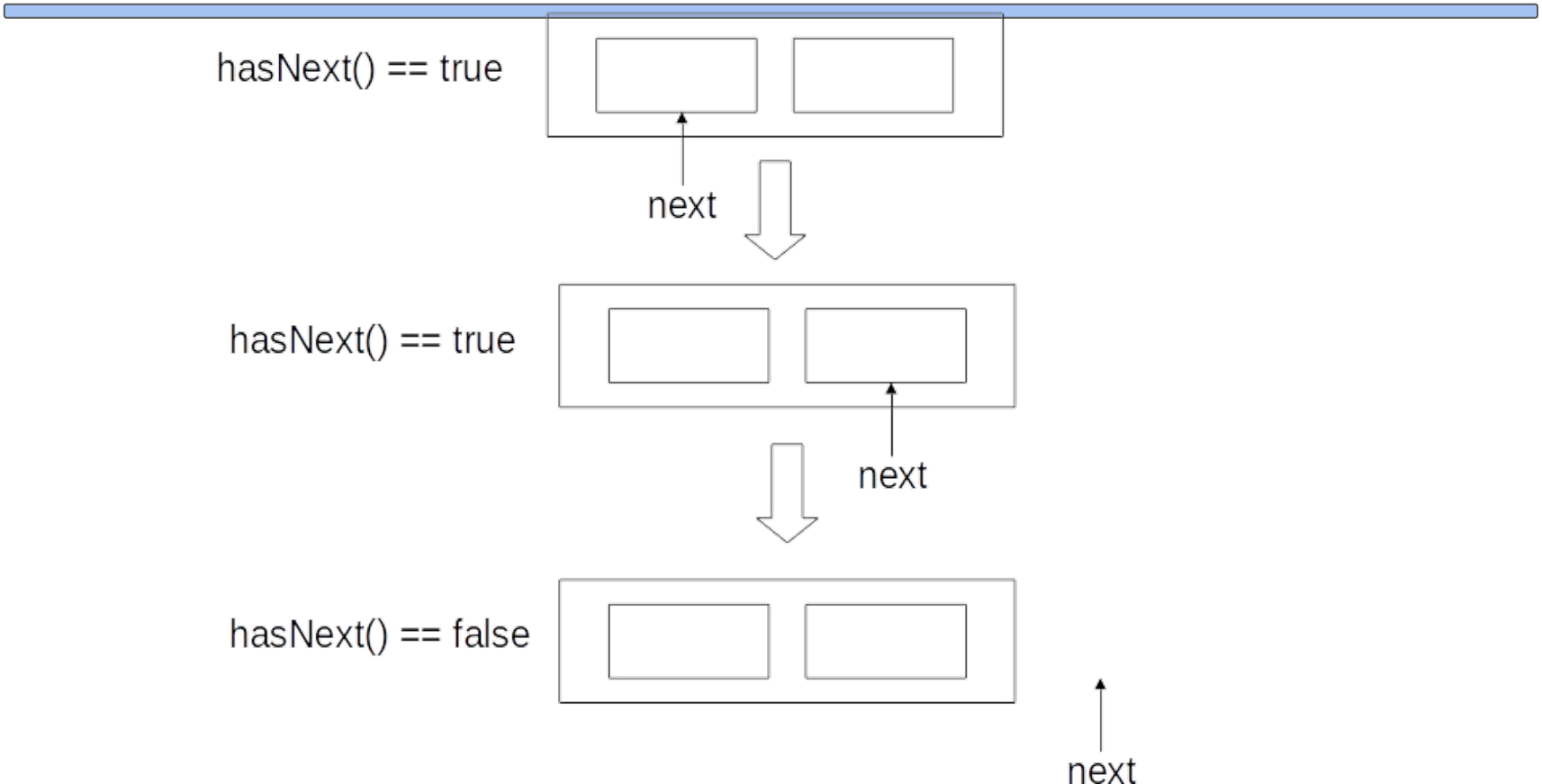
- Allows a user to step through each item in a data structure
  - array, vector, linked list, tree, etc.
- Two basic components
  - *hasNext()*
    - returns true if the iterator has any more items
  - *next()*
    - returns the next item in the iterator

Iterator

```
public static void main(String[] args) {
    List<String> kolekcija = new ArrayList<>();
    kolekcija.add("Vienas");
    kolekcija.add("Du");
    Iterator<String> iteratorius = kolekcija.iterator();

    while (iteratorius.hasNext()) {
        System.out.println(iteratorius.next());
    }
}
```

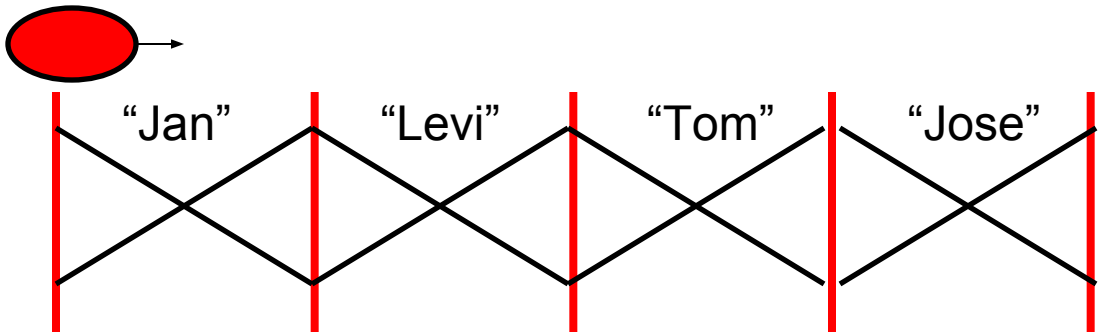
Iterator



Using an Iterator

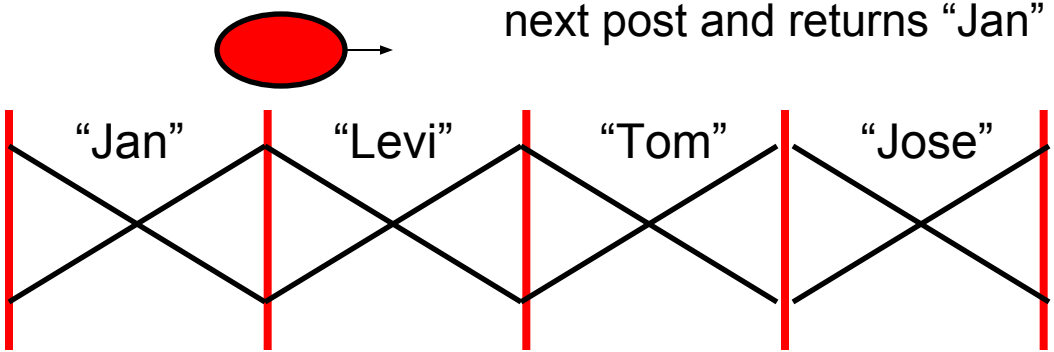
- When the iterator() method in a collection is invoked, it returns an “iterator object”
- We can then invoke the methods hasNext() and next() on that object, to iterate through the collection
    - (Those are the methods that are specified in the Iterator<T> interface)

```
ArrayList<String> names = new ArrayList<String>();
names.add("Jan");
names.add("Levi");
names.add("Tom");
names.add("Jose");
Iterator<String> it = names.iterator();
int i = 0;
```

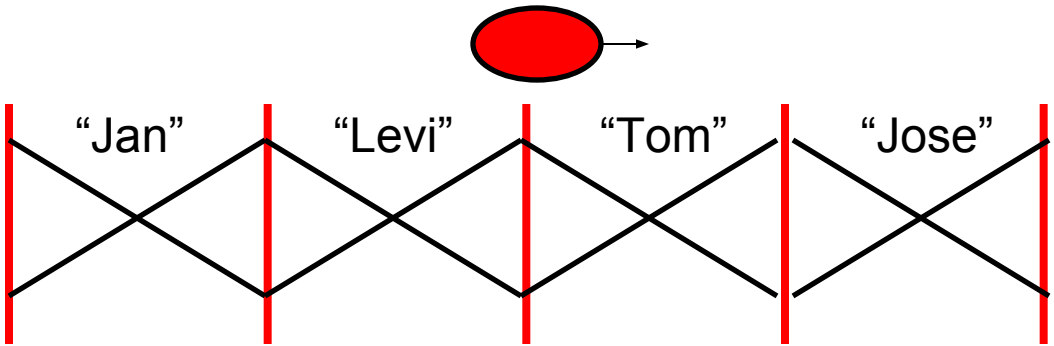


```
while( it.hasNext() ) {
    i++;
    System.out.println( it.next() );
}
// when i == 1, prints out Jan
```

first call to next moves iterator to next post and returns “Jan”

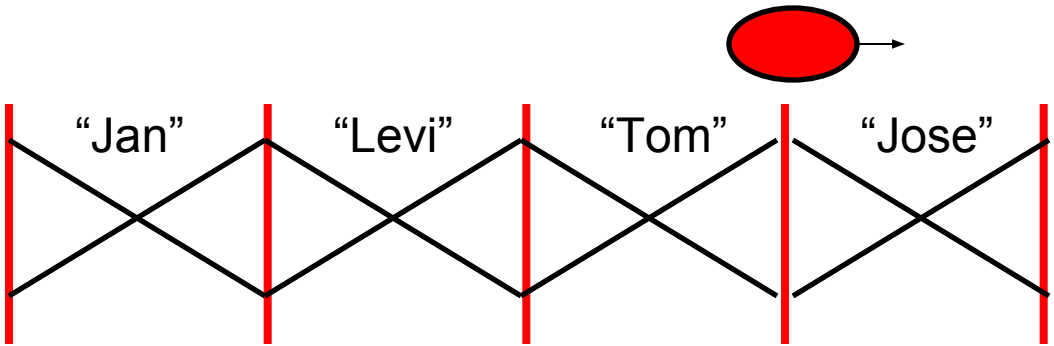


```
while( it.hasNext() ) {
      i++;
     System.out.println( it.next() );
}
// when i == 2, prints out Levi
```

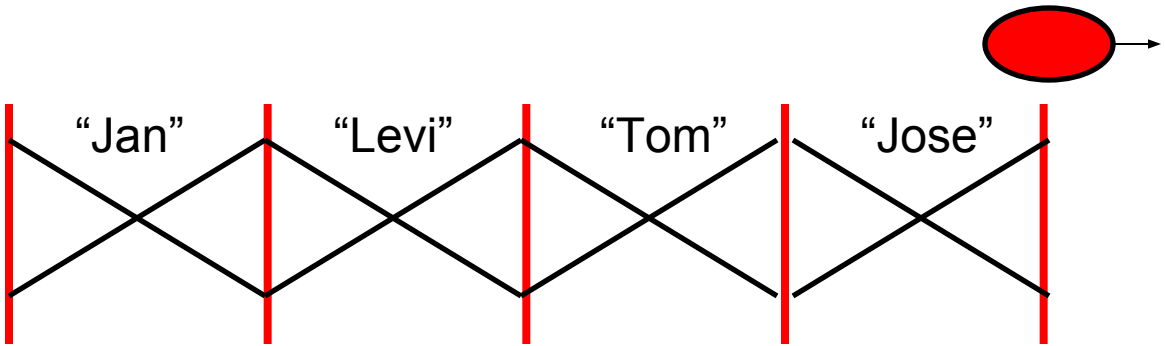


Fence Analogy

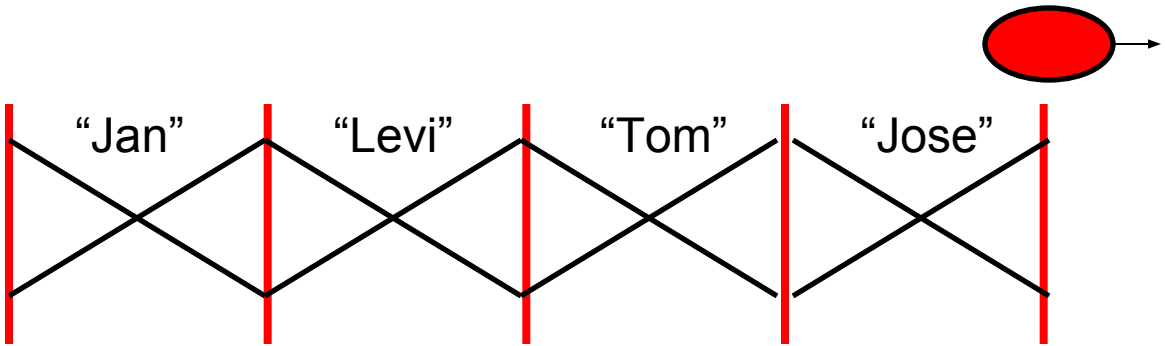
```
while( it.hasNext() ) {
      i++;
     System.out.println( it.next() );
}
// when i == 3, prints out Tom
```



```
while( it.hasNext() ) {
      i++;
     System.out.println( it.next() );
}
// when i == 4, prints out Jose
```



```
while( it.hasNext() ) {
     i++;
     System.out.println( it.next() );
}
// call to hasNext returns false
// while loop stops
```



Typical Iterator Pattern

```
public void printAll(Collection<String> list) {
    Iterator<String> it = list.iterator();
    while( it.hasNext() ) {
            String temp = it.next();
            System.out.println( temp );
        }
}
```

Question

8 What is output by the following code?

```
ArrayList<Integer> list;
list = new ArrayList<Integer>();
list.add(3);
list.add(3);
list.add(5);
Iterator<Integer> it = list.iterator();
System.out.print(it.next() + " ");
System.out.print(it.next() + " ");
System.out.print(it.next());
```

A. `3` B. `3 5` C. `3 3 5`

D. `3 3` E. `3 3` then a runtime error

remove method

- An `Iterator` can be used to remove things from the `Collection`
- Can only be called once per call to `next()`

```
public void removeWordsOfLength(int len) { Iterator<String> it =
   myList.iterator
    while( it.hasNext() ) {
      String temp = it.next();
       if(temp.length() == len)
        it.remove();
    }
}
// original list = [“dog”, “cat”, “hat”, “sat”]
// resulting list after removeWordsOfLength(3) ?
```

The Iterable Interface

A related interface is `Iterable`

One method in the interface:

```
public Iterator<T> iterator()
```

Why?

Anything that implements the `Iterable` interface can be used in the **for each** loop.

```
 ArrayList<Integer> list;
//code to create and fill list
int total = 0;
for( int x : list ){
    total += x;
 }
```

Iterable

If you simply want to go through all the elements of a Collection (or Iterable thing) use the for each loop

- hides creation of the Iterator

```
public void printAllOfLength(ArrayList<String> names,
                                                    int len){
    //pre: names != null, names only contains Strings
    //post: print out all elements of names equal in
    // length to len
     for(String s : names){
        if( s.length() == len ){
                System.out.println( s );
            }
     }
}
```

U5

Metodas kuris iššspausdintu visas argumento (Collection<String> kolekcija) reikšmes

Panaudoti **for each**

U6

Metodas, kuris išspausdintų visas argumento (Collection<String> kolekcija) reikšmes

Panaudoti .**iterator()**