Föreläsning 2

- Collection – List, Set och Queue
- Map
- Iterator, Iterable

Collection – ett interface

Collection är ett interface vilket definierar grundläggande funktionalitet i ett antal klasser i vilka man kan lagra godtyckliga objekt, sk *containerklasser / objektsamlingar*.

Method Summary	
boolean	**add**(E o) Ensures that this collection contains the specified element.
boolean	**addAll**(Collection<? extends E> c) Adds all of the elements in the specified collection to this collection.
void	**clear**() Removes all of the elements from this collection.
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.
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.
boolean	**removeAll**(Collection<?> c) Removes all this collection's elements that are also contained in the specified collection.
boolean	**retainAll**(Collection<?> c) Retains only the elements in this collection that are contained in the specified collection.
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.

Collection - samlingar

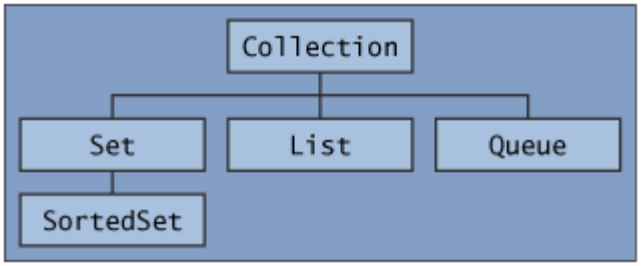
Ett interface A kan ärva ett annat interface B:

public interface A extends B { … }

Det innebär att en klass som implementerar interfacet A

public class AClass implements A {…}:

- ska implementera samtliga metoder i interfacen A och B
- även implementerar interfacet B
  B ref = new AClass();



Några interface ärver ***Collection*** för flexiblare funktionalitet:

- ***List*** - en ordnad sekvens av element. Ett element kan förekomma flera gånger i listan. Varje element kan nås med ett index.
- ***Set*** - en mängd där inga element förekommer mer än en gång.
- ***Queue*** - en kö innehåller en sekvens av element. I kön lagras och hämtas element normalt enligt FIFO-principen.

Containerklasser – konkreta klasser

Implementeringar av **List**, **Set** och **Queue** utgör tillsammans med implementeringar av interfacet **Map** en uppsättning containerklasser i Java, klasser man kan använda för att lagra objekt.

Tabellen visar ett urval av klasser som implementerar ***Set***, ***List***, ***Queue*** och ***Map***.

Interfaces	Implementations				
	Hash table	Resizable array	Tree	Linked list	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue				LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

I ovanstående tabell saknas Javas ursprungliga objektsamlingar, **Vector** och **Hashtable**. Dessa finns fortfarande i Java, framför allt av historiska skäl. Värt att notera är att klassen Vector implementerar Collection.

Interfacet List<E>

Gränssnittet **List<E>** innehåller, förutom metoderna i Collection, bl.a. metoderna:

Method Summary	
void	**add**(int index, E element) Inserts the specified element at the specified position in this list.
E	**get**(int index) Returns the element at the specified position in this list.
int	**indexOf**(Object o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
int	**lastIndexOf**(Object o) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
E	**remove**(int index) Removes the element at the specified position in this list
E	**set**(int index, E element) Replaces the element at the specified position in this list with the specified element.
List<E>	**subList**(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.

Klasser som implementerar **List** (och därmed Collection) är bl.a.

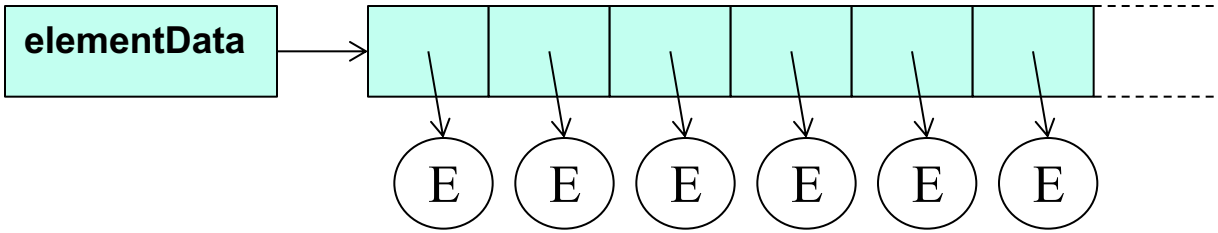
- **public class ArrayList<E> implements List<E> {…}**
- **public class LinkedList<E> implements List <E> {…}**
- **public class Vector<E> implements List<E> {…}**.

Den underliggande strukturen, *datastrukturen*, för att lagra elementen är dold för användaren. Klassernas namn ger dock information om datastrukturen som används.

List-implementationer

I klassen ***ArrayList<E>*** finner man instansvariabeln:

private transient E[] elementData;



Det innebär att listan implementeras med en array, vilket klassens namn antyder. Implementeringen i array innebär bl.a. att det går väldigt snabbt att komma åt element med hjälp av index.

Constructor Summary
`ArrayList()` Constructs an empty list with an initial capacity of ten.
`ArrayList(Collection<? extends E> c)` Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
`ArrayList(int initialCapacity)` Constructs an empty list with the specified initial capacity.

Klassen ***Vector<E>*** har en liknande implementation som *ArrayList* och likvärdiga egenskaper. Numera används *ArrayList* men *Vector* kan dyka upp i gammal kod.

ArrayListEx.java

List-implementationer

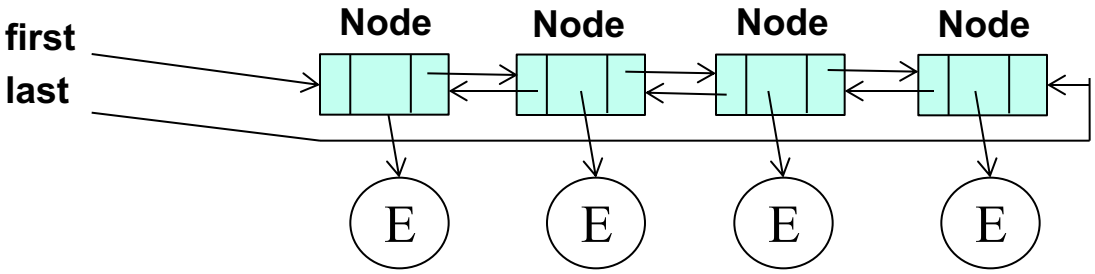
LinkedList<E> implements List<E>

I klassen ***LinkedList*** finner man instansvariablerna:

```
transient Node<E> first;
transient Node<E> last;
```

Node<E>
- prev: Node<E> - Item : E - next: Node<E>

Listan är implementerad som en dubbellänkad lista:



Element först och sist i listan hanteras snabbt.

Linkedlist implementerar dessutom metoder för att hantera Stack respektive Queue.

LinkedListEx.java | LinkedStack.java | UseLinkedStack.java

Collections – metoder för listor

I klassen ***Collections*** finns ett antal klassmetoder att använda tillsammans med List-implementationer, t.ex:

public static void sort(List<E> l)
Listan ordnas. Elementen i listan måste implementera Comparable.

public static void sort(List<E> l, Comparator<E> c)
Elementen i listan ordnas med Comparator-implementeringen.

public static int binarySearch(List<E> l, E element)
Effektiv sökning i lista ordnad med elementens Comparable-implementering.

public static int binarySearch(List<E> i, Comparator<E> c, E element)
Effektiv sökning i lista ordnad med Comparator-implementeringen

public static void reverse(List<E> l)
Elementens ordning blir omvänd

public static void rotate(List<E> l, int n)
Elementen flyttas n steg åt höger (n<0 åt vänster)

public static void shuffle(List<E> l)
Elementen i listan blandas slumpmässigt

public static void swap(List<E> l, int i, int j)
Byter plats på elementen i positionerna i och j

Collections – metoder för listor

Men innan en del av metoderna kan användas (sort, binarySearch) måste klassen *Person* implementera ***Comparable<Person>***.
Hur ser en implementering ut, vilken ordnar *Person*-objekt växande efter id ?

```
public class Person implements Comparable<Person> {
   :
   public int compareTo( Person p ) {
      return id.compareTo(p.id);      // p.id.compareTo(id) ordnar avtagande
   }
}
```

Ett alternativ är att skriva en klass vilken implementerar ***Comparator<Person>***.
Hur ser en klass ut, vilken implementerar *Comparator* på så sätt att *Person*-objekt ordnas avtagande efter id ?

```
public class IdDesc implements Comparator<Person> {
   public int compare( Person p1, Person p2 ) {
      return p2.getId().compareTo( p1.getId() );
   }
}
```

List1.java

ListSpeed.java

Queue-implementationer

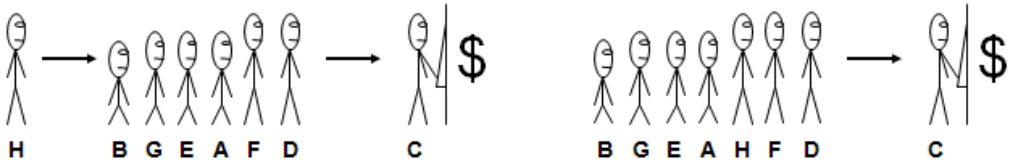
Interfacet ***Queue<E>*** innehåller 2 uppsättningar av tre metoder:

	Throws exception	*Returns special value*
Insert	`add(e)`	`offer(e)`
Remove	`remove()`	`poll()`
Examine	`element()`	`peek()`

Uppsättningen till vänster kastar ett undantag om ett fel uppstår medan uppsättningen till höger returnerar speciella värden.

Klassen ***LinkedList*** implementerar **Queue<E>**.

Klassen ***PriorityQueue*** implementerar **Queue<E>** på så sätt att element placeras i kön efter *prioritet*. Element med samma prioritet bildar en normal kö i prioritetskön.
Prioriteten ges av elementens *Comparable*-implementeringen eller av en separat klass som implementerar *Comparator*.



QueueEx.java

Map - avbildningar

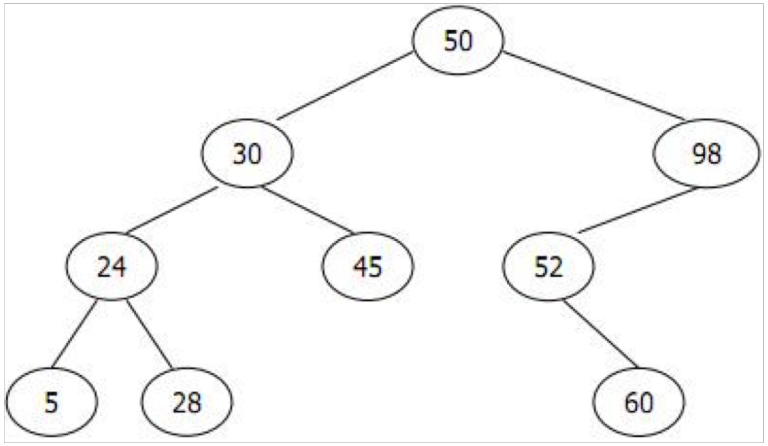
I en **Map** bildar en nyckel (*key*), och ett värde (*value*) ett par <*key,value*>. Med hjälp av nyckeln placeras värdet i objektsamlingen och med hjälp nyckeln söker man efter värdet.

Method Summary	
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.
V	**get**(Object key) Returns the value to which this map maps the specified key.
Set<K>	**keySet**() Returns a set view of the keys contained in this map.
V	**put**(K key, V value) Associates the specified value with the specified key in this map.
void	**putAll**(Map<? extends K,? extends V> t) Copies all of the mappings from the specified map to this map.
V	**remove**(Object key) Removes the mapping for this key from this map if it is present.
int	**size**() Returns the number of key-value mappings in this map.
Collection<V>	**values**() Returns a collection view of the values contained in this map.

Klasserna ***TreeMap<K,V>*** och ***HashMap<K,V>*** implementerar gränssnittet **Map<K,V>**.

Map - klasser

I en ***TreeMap*** lagras <key,value>-paret i en trädstruktur. Nyckeln avgör var i trädet paret lagras.



TreeMap håller nycklarna ordnande och medger snabb sökning.

HashMap håller ej nycklarna ordnade men medger mycket snabb sökning.

Maps.java

Search.java

Iterator

Gränssnittet Iterator innebär att en klass måste implementera tre metoder.

Method Summary	
boolean	**hasNext**() Returns true if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
void	**remove**() Removes from the underlying collection the last element returned by the iterator. Throws UnsupportedOperationException if the remove operation is not supported by this Iterator.

Med en Iterator går man i genom objekten i en containerklass:

```
ArrayList<Shape> shapes = new ArrayList<Shape>();
Shape shape;
:
Iterator<Shape> iter = shapes.iterator();
while( iter.hasNext() ) {
  shape = iter.next();
   :
}

for( Iterator<Shape> iter = shapes.iterator(); iter.hasNext(); ) {
  shape = iter.next();
   :
}
```

Iterable

Fr.o.m. Java 1.5 kan man iterera genom en containerklass med en förenklad for-loop. Men man kan inte ändra värde på enkla variabler eller byta referens i referensvariabler.

```
int[] numbers = { 11, 42, -13, 8, 14 };
int sum = 0;
for( int a : numbers )
  sum += a;

List<Double> list = new ArrayList<Double>();
for( int i = 0; i < 5; i++ )
  list.add( new Double( Math.random() ) );
for( Double nbr : list )
   System.out.println( nbr );
```

För att man ska kunna iterera genom en struktur måste strukturen implementera gränssnittet ***Iterable***.

Method Summary - Iterable	
Iterator<T>	**iterator**() Returns an iterator over a set of elements of type T.

Förenklad for-loop och objektsamlingar

För att den förenklade for-loopen ska kunna användas med en containerklass så måste klassen implementera gränssnittet Iterable. Då klassen implementerar Iterable lägger kompilatorn ut koden så här:

```
ObjectArray<Point> oa = new ObjectArray<Point>(); // ObjectArray implementerar Iterable
oa.add(new Point( 10, 2 ));
oa.add(new Point( 14, 9 ));
for(Point p : oa) {
   System.out.println( p );
   p.setLocation( 5, 9 );
}
```

Läggs ut som:

```
ObjectArray oa = new ObjectArray();
oa.add(new Point( 10, 2 ));
oa.add(new Point( 14, 9 ));
Point p;
for(Iterator i$ = oa.iterator(); i$.hasNext(); ) {
   p = (Point)i$.next();
   System.out.println( p );
   p.setLocation( 5, 9 )
}
```

Som du ser används först Iterable-implementeringen (**oa.iterator()**) och därefter Iterator-implementeringen (**i$.hasNext()** resp **i$.next()**). I for-loopen används en lokal variabel (**p**) med värde från aktuellt element.