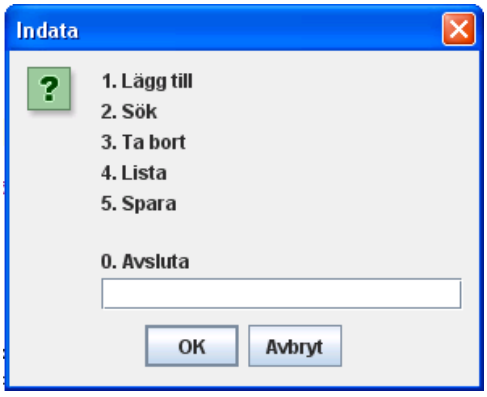
Laboration 13 – Hashtabell

Uppgifter

Uppgift 1-5 handlar om att använda ett HashMap-objekt, att hämta innehåll från fil, skriva innehåll till fil och lägga till / ta bort / söka efter objekt.
Packa upp laborationsfilerna i paketet ***laboration13***. ***lexikon.txt*** ska placeras i mappen ***files*** i projektkatalogen.

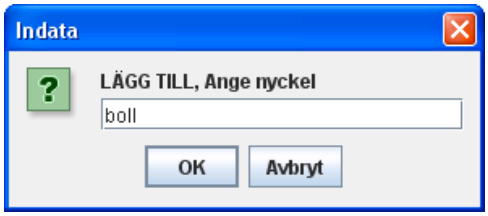
1. I metoden **example1()** i programmet **HashExample** skapas en *HashMap* och sedan anropas ett antal metoder. Förklara följande metoders funktion och vad de returnerar för värde (om de returnerar ett värde):
   * put (rad 12)
   * containsKey (rad 14)
   * containsValue (rad 15)
   * get (rad 16,17)
   * remove (rad 18,19)
   * size (rad 20)
   * isempty (rad 21)
   * clear (rad 34)

2a. Kör main-metoden i **Laboration13.java**. Följande händer:
   * programmet läser in information från *files/lexikon.txt* och placerar informationen i ett objekt av typen *HashMap*. Nycklarna utgörs av ord på svenska och värdena är motsvarande ord på engelska.
   * en programloop startas varvid meny-alternativen till höger visas. Just nu är det endast alternativen 0, 4 och 5 som fungerar. Välj alternativ 4 så ser du en listning över orden som lagras i hash-tabellen.



   Studera metoden **activity** så du förstår hur programmet fungerar.
   Studera även metoden **readFromFile** som visar hur man kan läsa från en textfil med angiven teckenkodning.

2b. När användaren väljer alternativ 1 ska följande hända:
   * En inmatningsdialog ska visa sig där användaren får mata in ett ord på svenska. Det inmatade ordet ska lagras i variabeln *key* (av typen String).



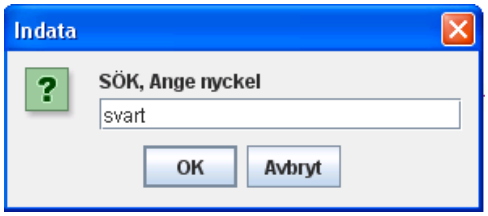
   * En inmatningsdialog ska visa sig där användaren får mata in ett ord på engelska. Det inmatade ordet ska lagras i variabeln *value* (av typen String).

   * Lägg till de inmatade värdena i hash-tabellen med *put*-metoden.

   Din uppgift är att komplettera metoden **add** med kod. Kontrollera resultatet genom anrop till alternativ 4 (Lista).

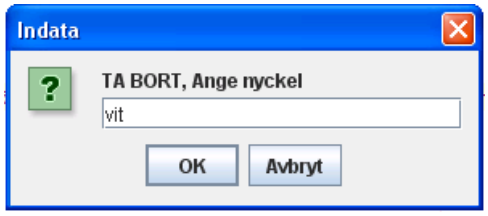
2c. När användaren väljer alternativ 2 ska följande hända:
* En inmatningsdialog ska visa sig där användaren får mata in ett ord på svenska.
* Sök efter det inmatade ordet i hash-tabellen. Resultatet av sökningen ska skrivas ut på formen:
  `nyckel=svart, värde=black`

Din uppgift är att komplettera metoden **seek** med kod.



2d. När användaren väljer alternativ 3 ska följande hända:
* En inmatningsdialog ska visa sig där användaren får mata in ett ord på svenska.
* Det inmatade ordet ska tas bort ur hash-tabellen. Borttagning sker endast om det inmatade ordet är en nyckel i hashtabellen.

Din uppgift är att komplettera metoden **remove** med kod.



3a. Klassen **StringValues** innehåller arrayen *lists*,
```
private LinkedList<String>[] lists.
```
Varje element i arrayen är ett LinkedList-objekt vilket kan lagra String-objekt.

Komplettera metoden
public void add(int index, String value)
med kod. Metoden ska lägga till *value* sist i ett av LinkedList-objekten i arrayen *lists*. *index* anger det LinkedList-objekt som *value* ska läggas till i. Om *index* är 0 ska *value* läggas till sist i det första LinkedList-objektet, om *index* är 1 ska *value* läggas till sist i det andra LinkedList-objektet osv.

När du är färdig med *add*-metoden ska main-metoden i klassen **StringValues** ge körresultatet:

```
0:
1: 5-stjärningt
2:
3: Magnifikt
4: Utmärkt
5: Fint, Kanon, Helgjutet
6:
7: Bra, Utsökt, Excellent, Superbt
8:
9: Gott
```

3b. I klassen **StringValues** ska du lägga till metoden
public boolean remove(int index, String value)
Metoden ska ta bort *value* ur LinkedList-objektet med angivet *index*. Metoden ska returnera *true* om en sträng tas bort och annars *false*.

Om du aktiverar de tre översta raderna som är bortkommenterade i *main*-metoden så ska du få följande körresultat när du är färdig med *remove*-metoden.

```
Tar bor 'Kanon' ur lista 3: false
Tar bor 'Kanon' ur lista 5: true
Tar bor 'Kanon' ur lista 5: false
   0:
   1: 5-stjärningt
   2:
   3: Magnifikt
   4: Utmärkt
   5: Fint, Helgjutet
   6:
   7: Bra, Utsökt, Excellent, Superbt
   8:
   9: Gott
```

3c. I klassen **StringValues** ska du lägga till metoden
public int size()
Metoden ska returnera antalet strängar som lagras i de olika LinkedList-objekten. Om du aktiverar den sista avmarkerade raden i main-metoden ska du få följande körresultat när du är färdig med size-metoden:

```
Tar bor 'Kanon' ur lista 3: false
Tar bor 'Kanon' ur lista 5: true
Tar bor 'Kanon' ur lista 5: false
Antal värden: 10
   0:
   1: 5-stjärningt
   2:
   3: Magnifikt
   4: Utmärkt
   5: Fint, Helgjutet
   6:
   7: Bra, Utsökt, Excellent, Superbt
   8:
   9: Gott
```

Uppgift 4 handlar om att komplettera klassen **HashtableCH** med kod. Testa metoderna successivt medan du färdigställer dem.

4a. Skriv metoden
public V get(K key)
vilken ska söka efter en nyckel i tabellen. Om nyckeln finns ska motsvarande värde returneras, annars ska *null* returneras (se F14).

4b. Skriv metoden
public V remove(K key)
vilken ska ta bort paret <*key,value*> ur tabellen om det finns där. *value* ska i så fall returneras, annars ska *null* returneras.

4c. Skriv metoden
public int size()
vilken ska returnera antalet <*key,value*>-par som lagras i tabellen.

4d. Skriv metoden
public boolean isEmpty()
vilken ska returnera *true* om inga <*key,value*>-par lagras i tabellen och annars *false*.

4e. Skriv metoden
public boolean containsKey(K key)
vilken ska returnera *true* om *key* lagras i tabellen och annars *false*

4f. Skriv metoden
public V clear()
vilken ska tömma tabellen på element, dvs ändra värdena i **Bucket**-objekten så att instansvariablerna är *null* och tillståndet *EMPTY*. Glöm inte nollställa size!

4g. Skriv metoden
public Iterator<V> values()
vilken ska returnera ett objekt som implementerar Iterator<V> (se F14, keys)

Uppgift 5 handlar om att komplettera klassen **HashtableOH** med kod. Testa metoderna successivt medan du färdigställer dem.

5a. Skriv metoden
public V get(K key)
vilken ska söka efter en nyckel i tabellen. Om nyckeln finns ska motsvarande värde returneras, annars ska *null* returneras (se F14).

5b. Skriv metoden
public V remove(K key)
vilken ska ta bort paret <*key,value*> ur tabellen om det finns där. *value* ska i så fall returneras, annars ska *null* returneras.

5c. Skriv metoden
public int size()
vilken ska returnera antalet <*key,value*>-par som lagras i tabellen.

5d. Skriv metoden
public boolean isEmpty()
vilken ska returnera *true* om inga <*key,value*>-par lagras i tabellen och annars *false*.

5e. Skriv metoden
public boolean containsKey(K key)
vilken ska returnera *true* om *key* lagras i tabellen och annars *false*

5f. Skriv metoden
public V clear()
vilken ska tömma tabellen på element, dvs. anropa *clear*-metoden för samtliga *LinkedList*-objekt.

5g. Skriv metoden
public Iterator<K> keys()
vilken ska returnera ett objekt som implementerar Iterator<K> (se F14, keys)

5h. Skriv metoden
public Iterator<V> values()
vilken ska returnera ett objekt som implementerar Iterator<V> (se F14, keys)

Extra

6. Förbättra **grow**-metoden i HashtableCH så att Bucket-objekten i den gamla tabellen återanvänds i den nya. Ett sätt kan vara att först flytta de Bucket-objekt som innehåller <key, value>-par och sedan flytta de som är tomma till lediga positioner i den nya tabellen. Slutligen ska du se till så att alla positioner i den nya tabellen innehåller Bucket-objekt.

7. Ändra **keys**-metoden i **HashtableCH** så att metoden returnerar ett **KeyIterator**-objekt:

```
public Iterator<K> keys() {
    return new KeyIterator();
}
```

Klassen *KeyIterator* ska vara en inre klass i *HashtableCH* vilken implementerar *Iterator<K>*:

```
private class KeyIterator implements Iterator<K> {
    :
}
```

8. Ändra **values**-metoden i klassen **HashtableCH** så att metoden returnerar ett **ValueIterator**-objekt:

```
public Iterator<V> values() {
    return new ValueIterator();
}
```

Klassen *ValueIterator* ska vara en inre klass i *HashtableCH* vilken implementerar *Iterator<V>*. Internt ska klassen använda ett objekt av typen *KeyIterator*:

```
private class ValueIterator implements Iterator<V> {
    private Iterator<K> keys = keys();
    :
}
```

9. Förändra **HashtableOH** så att arrayen fördubblar sin storlek när antalet element som lagras i tabellen uppgår till 75% av arrayens kapacitet. Din lösning kan delvis likna den tidiga versionen i *HashtableCH*.
En ytterligare förbättring kan vara att återanvända *Entry*-objekten och att *LinkedList*-objekten från den gamla tabellen.

Lösningar

Uppgift 2b

```
private void add( HashMap<String,String> map ) {
    String key = JOptionPane.showInputDialog( "LÄGG TILL, Ange nyckel" );
    String value = JOptionPane.showInputDialog( "LÄGG TILL, Ange värde" );
    map.put( key, value );
}
```

Uppgift 2c

```
private void seek( HashMap<String,String> map ) {
    String key = JOptionPane.showInputDialog( "SÖK, Ange nyckel" );
    String value = ( String )map.get( key );
    System.out.println( "nyckel=" + key + ", värde=" + value );
}
```

Uppgift 2d

```
private void remove( HashMap<String,String> map ) {
    String key = JOptionPane.showInputDialog( "TA BORT, Ange nyckel" );
    map.remove( key );
}
```

Uppgift 3a

```
public void add( int index, String value ) {
    lists[index].add(value);
}
```

Uppgift 3b

```
public boolean remove( int listIndex, String value ) {
    for( int elemIndex=0; elemIndex<lists[listIndex].size(); elemIndex++)
        if(lists[listIndex].get(elemIndex).equals(value)) {
            lists[listIndex].remove(elemIndex);
            return true;
        }
    return false;
}
```

eller

```
public boolean remove( int listIndex, String value ) {
    int index = lists[listIndex].indexOf(value);
    if(index>=0) {
        lists[listIndex].remove(index);
        return true;
    }
    return false;
}
```

Uppgift 3c

```
public int size() {
    int count=0;
    for( LinkedList<String> valueList : lists ) {
        count += valueList.size();
    }
    return count;
}
```

Uppgift 4a - 4g

```
public V get( K key ) {
    int hashIndex = hashIndex( key );
    while( ( table[ hashIndex ].state != Bucket.EMPTY ) && !key.equals(
table[ hashIndex ].key ) ) {
        hashIndex++;
        if(hashIndex==table.length)
            hashIndex=0;
    }
    if( table[ hashIndex ].state == Bucket.OCCUPIED )
        return table[ hashIndex ].value;
    return null;
}

public V remove( K key ) {
    V res = null;
    int hashIndex = hashIndex( key );
    while( ( table[ hashIndex ].state != Bucket.EMPTY ) && !key.equals(
table[ hashIndex ].key ) ) {
        hashIndex++;
        if(hashIndex==table.length)
            hashIndex=0;
    }
    if( table[ hashIndex ].state == Bucket.OCCUPIED ) {
        res = table[ hashIndex ].value;
        table[ hashIndex ].key = null;
        table[ hashIndex ].value = null;
        table[ hashIndex ].state = Bucket.REMOVED;
        size--;
    }
    return res;
}

public int size() {
    return size;
}

public boolean isEmpty() {
    return size==0;
}

public boolean containsKey(K key) {
    return get(key)!=null;
}

public void clear() {
    for(int index=0; index<table.length; index++) {
        if(table[ index ].state != Bucket.EMPTY) {
            table[ index ].key = null;
            table[ index ].value = null;
            table[ index ].state = Bucket.EMPTY;
        }
    }
    size = 0;
}

public Iterator<V> values() {
    ArrayList<V> values = new ArrayList<V>();
    for(int i=0; i<table.length; i++)
        if( table[ i ].state == Bucket.OCCUPIED )
            values.add(table[ i ].value);
    return values.iterator();
}
```

Uppgift 6

```
private void grow() {
    Bucket<K,V>[] oldTable = table;
    table = (Bucket<K,V>[])new Bucket[ table.length*2 ];
    size = 0;
    threshold = (int)(loadfactor*table.length);
    for(int index=0; index<oldTable.length; index++) {
        if( oldTable[index].state == Bucket.OCCUPIED )
            put( oldTable[index] );
    }
    int startIndex = 0;
    for(int index=0; index<oldTable.length; index++) {
        if( oldTable[index].state != Bucket.OCCUPIED )
            startIndex = put( oldTable[index], startIndex );
    }
    while( startIndex < table.length ) {
        if( table[ startIndex ] == null )
            table[ startIndex ] = new Bucket<K,V>();
        startIndex++;
    }
}

private void put( Bucket<K,V> bucket ) {
    int hashIndex = hashIndex( bucket.key );
    while( table[ hashIndex ] != null ) {
        hashIndex++;
        if(hashIndex==table.length)
            hashIndex=0;
    }
    table[ hashIndex ] = bucket;
    size++;
}

private int put(Bucket<K,V> bucket, int index) {
    bucket.state = Bucket.EMPTY;
    while( table[ index ] != null ) {
        index++;
        if(index==table.length)
            index=0;
    }
    table[ index ] = bucket;
    return index+1;
}
```

Uppgift 7

```
private class KeyIterator implements Iterator<K> {
    private ArrayList<K> keys = new ArrayList<K>();
    private int listIndex = 0;

    public KeyIterator() {
        for(int i=0; i<table.length; i++)
            if( table[ i ].state == Bucket.OCCUPIED )
                keys.add(table[ i ].key);
    }

    public boolean hasNext() {
        return listIndex < keys.size();
    }

    public K next() {
        return keys.get(listIndex++);
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

Uppgift 8

```
private class ValueIterator implements Iterator<V> {
    Iterator<K> iter = keys();

    public boolean hasNext() {
        return iter.hasNext();
    }

    public V next() {
        return get(iter.next());
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```