java.util

Collections pakkinn í java.util inniheldur margar góðar gagnagrindur

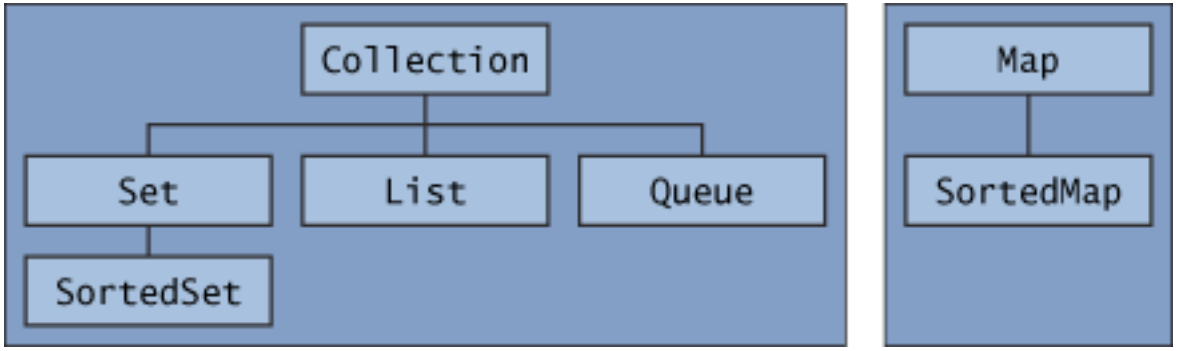
- Allar mjög góðar, erfitt að búa til eitthvað betra
- Mjög mikið notaðar
- Halda utan um gögn á skipulegan hátt, rétt eins og fylki og Vector

Lýsingar á gagnagrindum í gegnum skil

- Auðvelt að skipta um gagnagrind seinna
- Skrifum kóða sem tekur sem inntak eins víð skil og mögulegt er
  - ekki biðja um Vektor sem inntak ef að List myndi duga
  - Fyrir inntak er oftast nóg að fá Iterator

Förum yfir 3 tegundir af skilum

- List – safn hluta í röð, eins og fylki
- Set – safn hluta, eins og mengi, engin sérstök röð, hver hlutur kemur fyrir einu sinni
- Map – einnig kallað Dictionary/Associative array/Table geymir vörpun frá lyklum yfir í gildi.



List<E>

List<E> geymir hluti af taginu E í röð

- Vector<E> og LinkedList<E> útfæra List<E> skilin
- Algengar aðferðir
  - E get(int i) – skilar staki í sæti i
  - E set(int i, E e) – setur e í sæti i (skilar því sem var fyrir)
  - add(E e) – bætir e við aftast
  - add(int i, E e) – setur e í sæti i og hliðrar því sem er fyrir aftan
  - E remove(int i) – tekur sæti i út og skilar gildinu

Dæmi:

```
List<String> v = new Vector<String>();
for (int i = 0; i < args.length; i++){
   v.add(args[i]);
}
v.set(0,"Hello");
v.add(0,"World");
v.remove(1); // Tekur út "Hello"
```

Collections

Collections klasinn inniheldur mörg reiknirit fyrir List

- sort – raðar lista í “rétta” röð
- shuffle – umraðar lista í slembiröð
- swap – skiptir á tveimur gildum í lista
- binarySearch – hraðvirk leit í röðuðum lista

```
List<Integer> v;
Collections.sort(v); // Integer er comparable

int i = v.binarySearch(v,new Integer(0));
```

Öðruvísi röð

Hvernig röðum við í öðruvísi röð?

- Collections.sort tekur líka inn Comparator
- Comparator eru skil sem hafa compare(E e1, E e2) aðferð rétt eins og compareTo í Comparable

```
public class MyCmp implements Comparator<Integer> {
   public int compare(Integer a, Integer b) {
      int x = a.getValue(), y = b.getValue();
      if (x%2==0) {
         if (y%2==0) return x-y;
         else return 1;
      } else {
         if (y%2==0) return -1;
         else return x-y;
      }
}

Collections.sort(v,new MyCmp());
```

Set<E> - ekki farið yfir í fyrirlestri

Geymir mengi af hlutum af taginu E

- Aðeins hægt að setja hvern hlut einu sinni
- Hraðvirkt að spurja um hlut og bæta hlutum við
- Helstu aðferðir
  - boolean contains(E e) – er e í menginu?
  - boolean add(E e) – bætum e við í mengið skilar true ef það var ekki fyrir
  - boolean remove(E e) – tekur e úr menginu, skilar true ef það var þar fyrir

Set<E> dæmi- ekki farið yfir í fyrirlestri

```
Set<String> s = new HashSet<String>();
for (String a : args) {
   s.add(a);
}

System.out.println(s.size());
for (String a : s) {
   System.out.println(a);
}
```

```
% java test a a a b c d d b a
4
d
b
c
a
```

Set<E> dæmi- ekki farið yfir í fyrirlestri

```
Set<String> s = new TreeSet<String>();
for (String a : args) {
   s.add(a);
}

System.out.println(s.size());
for (String a : s) {
   System.out.println(a);
}
```

```
% java test a a a b c d d b a
4
a
b
c
d
```

Map<K,V>- ekki farið yfir í fyrirlestri

Map<K,V> varpar hlutum tagi K yfir í hluti af taginu V. K er fyrir lyklana og V er fyrir gildin.

- Mjög öflug gagnagrind
- Leita að lyklum, bæta við gildum og breyta er allt hraðvirkar aðgerðir
- helstu aðferðir
  - V put(K key, V value) – setur gildið value fyrir key, skilar gamla gildinu fyrir key
  - V get(K key) – skilar gildinu fyrir key
  - V remove(K key) – eyðir gildinu fyrir key
  - boolean containsKey(K key) – er lykillinn key með gildi?

Map<K,V>- ekki farið yfir í fyrirlestri

Map<K,V> varpar hlutum tagi K yfir í hluti af taginu V. K er fyrir lyklana og V er fyrir gildin.

- Mjög öflug gagnagrind
- Leita að lyklum, bæta við gildum og breyta er allt hraðvirkar aðgerðir
- helstu aðferðir
  - V put(K key, V value) – setur gildið value fyrir key, skilar gamla gildinu fyrir key
  - V get(K key) – skilar gildinu fyrir key
  - V remove(K key) – eyðir gildinu fyrir key
  - boolean containsKey(K key) – er lykillinn key með gildi?

Aldrei nota Map með breytanlegum hlutum! (eða a.m.k. ekki breyta þeim)

Map<K,V> dæmi- ekki farið yfir í fyrirlestri

Lesum inn orðabók á forminu

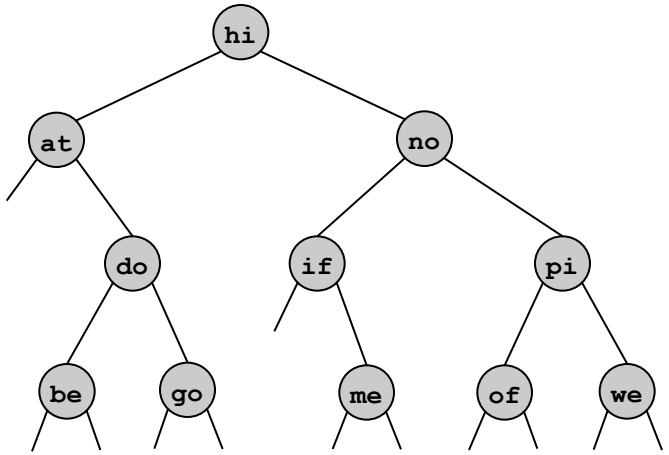
orð þýðing1 þýðing2 …

```
Map<String,List<String>> m =
      new HashMap<List<String>>();
while(!StdIn.isEmpty()) {
   String[] a = StdIn.readLine().split(" ");
   String k = a[0];
   m.set(k,new Vector<String>());
   for (int i = 1; i < a.length; i++) {
       m.get(k).add(a[i]);
   }
}
```

BST – Inorder iterator

Inorder röð

- Heimsækjum vinstra tré endurkvæmt
- Heimsækjum rótina
- Heimsækjum hægra tré endurkvæmt




```
inorder: at be do go hi if me no of pi we
```

```
public inorder() { inorder(root); }

private void inorder(Node x) {
   if (x == null) return;
   inorder(x.left);
   StdOut.println(x.key);
   inorder(x.right);
}
```

For-each lykkja með BST

For-each lykkja. Leyfir kóða að fara í gegnum BST í “réttri” röð, án þessu að vita hvernig BST er útfært.

```
ST<String, Integer> st = new BST<String, Integer>();
…

for (String s : st) {
   StdOut.println(st.get(s) + " " + s);
}
```

For-each lykkjan kallar á `st.iterator()` fallið sem skilar Iterator hlut. BST verður að útfæra `Iterable<Key>` skilin til að Java leyfi þetta.

For-each lykkja með BST

BST. Bætum við kóða til að skila iterator hlut

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class BST<Key extends Comparable<Key>, Value> implements Iterable<Key>  {

   private Node root;

   private class Node { … }

   public void put(Key key, Value val) { … }
   public Value get(Key key)           { … }
   public boolean contains(Key key)    { … }

   public Iterator<Key> iterator() { return new Inorder(); }
   private class Inorder implements Iterator<Key> {
       Stack<Node> stack = new Stack<Node>();
       Inorder() { pushLeft(root); }
       public void remove()     { throw new UnsupportedOperationException(); }
       public boolean hasNext() { return !stack.isEmpty();                   }
       public Key next() {
          if (!hasNext()) throw new NoSuchElementException();
          Node x = stack.pop();
          pushLeft(x.right);
          return x.key;
       }
       public void pushLeft(Node x) {
          while (x != null) {
             stack.push(x);
             x = x.left;
          }
       }
   }
}
```

For-each lykkja með BST

```
public class BST<Key extends Comparable<Key>, Value> implements Iterable<Key>
{

   private Node root;
   ...
   public Iterator<Key> iterator() { return new Inorder(); }
   private class Inorder implements Iterator<Key> {
       Stack<Node> stack = new Stack<Node>();
       Inorder() { pushLeft(root); }
       public void remove()      { throw new UnsupportedOperationException(); }
       public boolean hasNext() { return !stack.isEmpty();                    }
       public Key next() {
          if (!hasNext()) throw new NoSuchElementException();
          Node x = stack.pop();
          pushLeft(x.right);
          return x.key;
       }
       public void pushLeft(Node x) {
          while (x != null) {
             stack.push(x);
             x = x.left;
          }
       }
   }
}
```

Samhliða/samtíma vinnsla

- Stýrikerfi sér til þess að forrit keyri samtímis (samskeiða, concurrent)
- Ef > 1 örgjörvi getum við líka keyrt samhliða (parallel)
- Stýrikerfi skiptir um forrit sem á að keyra oft (ca. 20 ms)
  - sum forrit hafa lítið að gera, skipt sjaldnar
  - önnur sjá um mikilvæga hluti, keyra oftar
  - við höfum takmarkaða stjórn á þessu

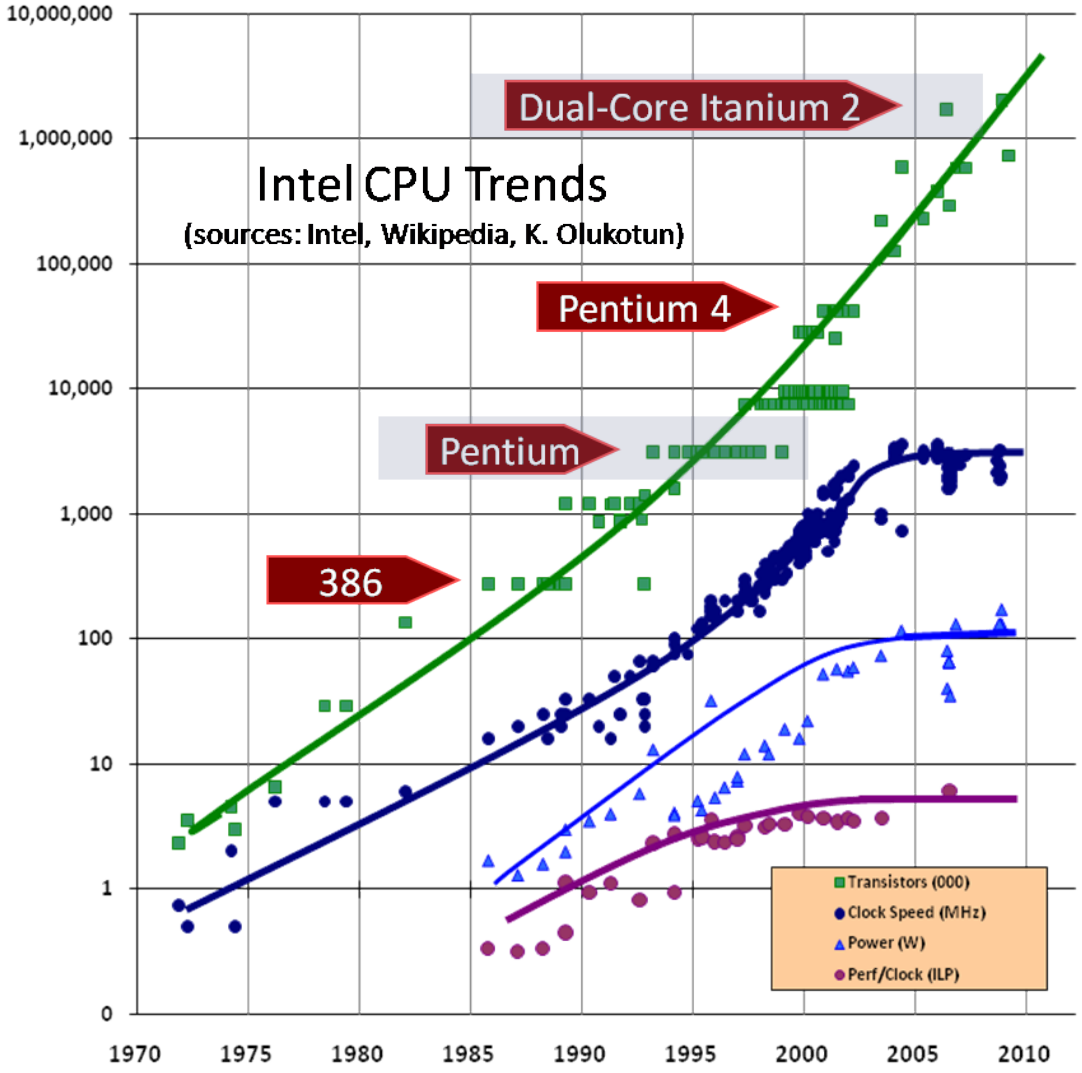
Samhliða vinnsla

Öll forrit keyra í “process”

- Stýrikerfi aðskilur forrit og einangrar þau hvort frá öðru
- Getum ekki kíkt á minnissvæði í öðrum forritum

- Þægileg blekking
  - Ef annað forrit krassar tökum við ekki eftir því
  - Einfaldar forritun, þurfum ekki að vita hvað er verið að keyra
  - Kostar sitt að halda utan um hvert forrit (ekki takmarkandi þáttur lengur)
  - Erfitt að tala við önnur forrit

Örgjörvahraði



The Free Lunch is Over – Herb Sutter
http://www.gotw.ca/publications/concurrency-ddj.htm

Samhliða vinnsla

Til að auka hraða og minnka viðbragðstíma brjótum við verkefni í forriti niður í smærri búta sem keyra samtímis.

Í Java (og öðrum forritunarmálum) er þetta gert með þráðum

- Hvert forrit getur haft marga þræði (þó takmarkað)
- Stýrikerfi ákveður hvaða þráð á að keyra
- Forrit getur líka haft áhrif á hvaða þráður er keyrður

Þræðir

- Keyra saman í forriti
- Hafa aðgang að sameiginlegum gögnum í minni, bæði les og skrif
- Geta samstillt sig (synchronize)
  - Geta beðið eftir að annar þráður klári keyrslu
  - Passað sig að skemma ekki fyrir öðrum þráðum

Samhliða vinnsla er erfið þegar stjórna þarf mörgum þráðum sem vinna með sameiginleg gögn

Þræðir í Java

Búum til klasa sem útfærir Runnable skil

```
public class C implements Runnable {
  …// smiður og tilviksbreytur
  public void run() {
     … // “Keyrum klasann”
  }
}
```

Búum til þráð sem keyrir C

```
C c = new C(…);
Thread t = new Thread(c);
t.start(); // keyrir run() aðferð
… // restin af forritinu keyrir samtímis
```

Counter

```
public class ThreadEx1 {
 private int count, N;
 public void add() {
     this.count++;
   }

 public ThreadEx1(int N) {
   this.N = N; this.count = 0;

   Thread t1 = new Thread(new Adder());
   Thread t2 = new Thread(new Adder());
   t1.start(); t2.start();
   try {
       t1.join();
       t2.join();
   }
   catch (InterruptedException e) {}
   System.out.println(this.count);
 }
}
```

```
private class Adder
    implements Runnable {
   public void run() {
       for (int i = 0; i < N; i++) {
         add();
       }
   }
}
```

```
public static void
    main(String[] args)  {
    ThreadEx1 te =
       new ThreadEx1(
          Integer.parseInt(args[0])
       );
    }
```

Counter

Virkar stundum :(

add() aðferðin breytir count tilviksbreytunni

- count++ er í raun 3 aðgerðir
- sama og count = count + 1
  1. ná í gildið á count
  2. leggja 1 við gildið
  3. geyma útkomuna í count
- Tveir þræðir gætu framkvæmt aðgerðina á sama tíma
- Fyrsti les count, leggur 1 við
- Annar les count
- Á sama tíma skrifar fyrsti þráður í count breytuna
- Annar þráður leggur 1 við og skrifar vitlaust gildi í count

Counter

Lausnin er að nota synchronized lykilorðið

- Tryggir að aðeins einn þráður keyri aðferð í einu
- Allir aðrir þræðir verða að bíða

```
public void add() {
     this.count++;
}
```

verður

```
public synchronized void add() {
     this.count++;
}
```

Counter

Einföld lausn, hætta að skrifa **alltaf** í sömu breytu

Hver þráður hefur sína breytu sem hann safnar upplýsingum í (i.e. count breytu)

Í lok for lykkju er sameiginlega breytan uppfærð með kalli á synchronized aðferð

Þessi hugmynd virkar mjög vel fyrr mörg fylkjaverkefni

- Finna stærsta/minnsta gildi í fylki
- Meðaltal fylkja (finna summu og telja)
- map-reduce

Counter

```
public class ThreadEx3 {
 private int count, N;
 public void
   synchronized add(int x) {
     this.count += x;
   }

 public ThreadEx3(int N) {
   this.N = N; this.count = 0;

   Thread t1 = new Thread(new Adder());
   Thread t2 = new Thread(new Adder());
   t1.start(); t2.start();
   try {
       t1.join();
       t2.join();
   }
   catch (InterruptedException e) {}
   System.out.println(this.count);
 }
}
```

```
private class Adder
    implements Runnable {
   public void run() {
       int count = 0;
       for (int i = 0; i < N; i++) {
          count++; // local breyta
       }
       add(count)
   }
}
```

```
public static void
    main(String[] args)  {
    ThreadEx3 te =
       new ThreadEx3(
          Integer.parseInt(args[0])
       );
    }
```

Lögmál Amdahls

Sum verkefni er hægt að leysa hraðar með þráðum (með mörgum örgjörvum).

Ef tíminn er T á einum örgjörva þá gæti hann í besta falli verið T/N á N örgjörvum, þ.e. N-föld hröðun

- Auka kostnaður í tíma við að forrita með þráðum
- Ekki alltaf hægt að keyra allt samhliða
- Þræðir eru alltaf að vinna á fullu, rekast aldrei hvor á annan

Lögmál Amdahls:

Ef hægt er að hraða P hluta af forriti S-falt verður heildar hröðun

$$\frac{1}{(1-P)+\frac{P}{S}}$$

Röðun

Mergesort má hraða með samhliða forritun

1. Skiptum fylkinu í tvo jafnstóra hluta
2. Röðum hlutunum endurkvæmt
3. Skeytum saman tveimur röðuðum fylkjum

Lið 2 má keyra samhliða. Lið 3 er einfaldast að keyra í einum þræði

- Búum til tvo þræði
  - Hver þráður geymir tilvísun á hlutfylki
  - Raðar hlutfylkinu
- Einn þráður bíður eftir hinum tveimur þráðunum
- Skeytir saman röðuðu fylkjunum

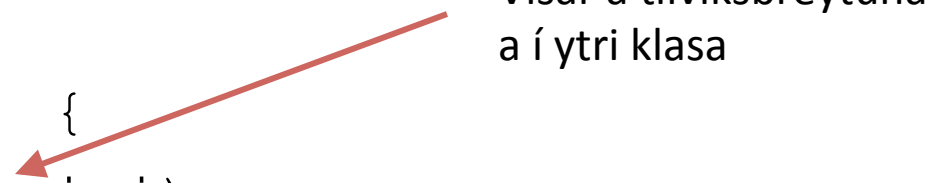
Parallel sort

```
import java.util.Arrays;

public class ParallelSort {
  private int[] a;
  public ParallelSort(int[] a) {
   this.a = a;
  }
  public void sort() { … }

  private class Sorter implements Runnable { … }

  public static void main(String[] args)  {
      int N = Integer.parseInt(args[0]);
      …
      ParallelSort PS = new ParallelSort(a);
      PS.sort();
  }
}
```

Parallel sort

Innri klasi sem raðar hlutfylki

```
private class Sorter implements Runnable {
   private int i,j;
   public Sorter(int i, int j) {
        this.i = i; this.j = j;
   }

   public void run() {
        Arrays.sort(a,i,j);
   }
}
```



Parallel sort

sort aðferðin

Skilgreinir hlutfylkin

```
public void sort() {
int N = a.length;
   int mid = N/2;
   Thread t1 = new Thread(new Sorter(0,mid));
   Thread t2 = new Thread(new Sorter(mid,N));
   t1.start(); t2.start();
   try {t1.join(); t2.join();} catch (InterruptedException e) {}
   // merge arrays
   int[] aux = new int[N]; int i = 0, j = mid;
   for (int k = 0; k < N; k++) {
       if (i == mid) aux[k] = a[j++];
       else if (j == N) aux[k] = a[i++];
       else if (a[j] < a[i]) aux[k] = a[j++];
       else aux[k] = a[i++];
   }
}
```

Sama "merge" og í mergesort í bók

Parallel sort

Skv. lögmáli Amdahls getum við búist við í mesta lagi 2x hröðun.

Endurkvæma kallið tekur O(N log N) tíma, merge hlutinn tekur O(N).

Ættum að sjá nálægt 2x hröðun fyrir stór fylki