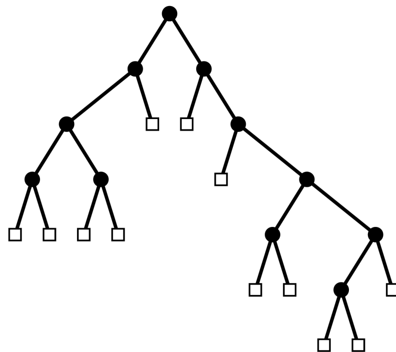
BST útfærsla

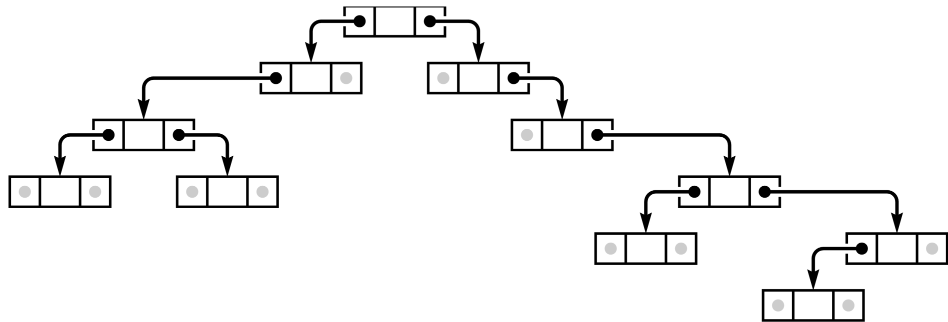
Notum tvær tilvísanir í hverjum hnút

Hver hnútur hefur

- lykil
- gildi
- tilvísun á vinstra tré
- tilvísun á hægra tré

```
private class Node {
    private Key key;
    private Value val;
    private Node left;
    private Node right;
}
```





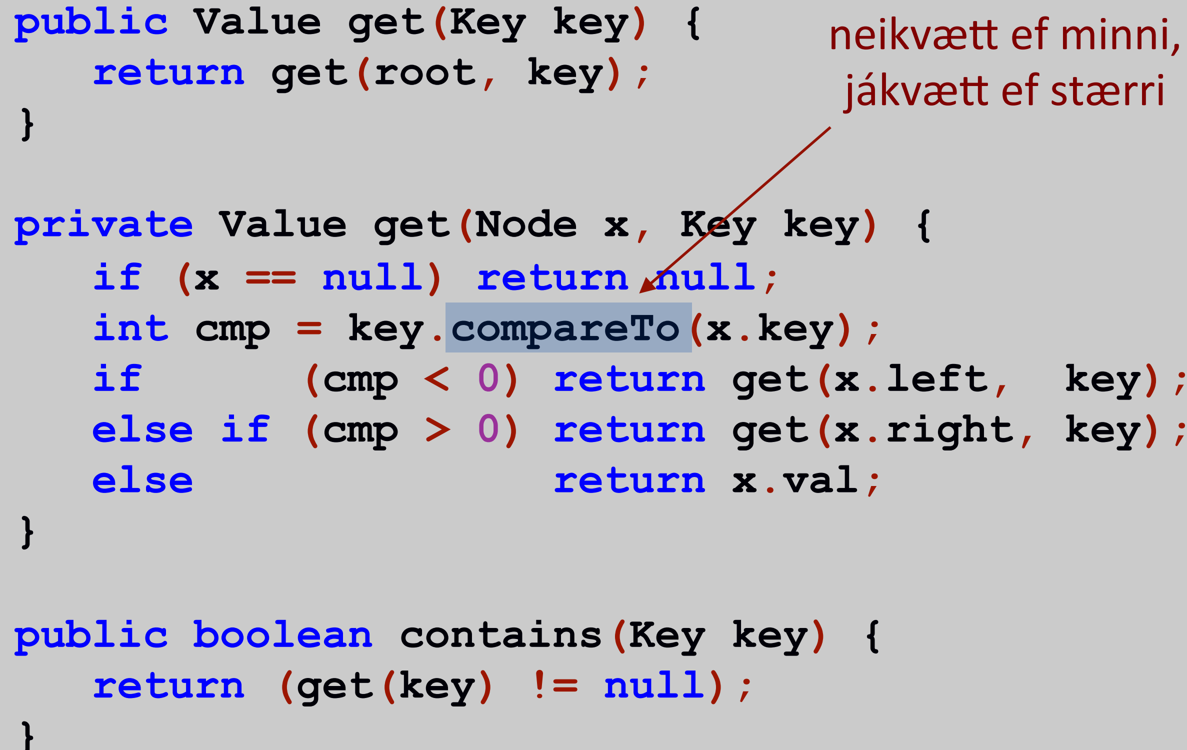
BST útfærsla

Upphafsstillt sem null

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

    private Node root;    // root of the BST

    private class Node {
        private Key key;
        private Value val;
        private Node left, right;

        private Node(Key key, Value val) {
            this.key = key;
            this.val = val;
        }
    }

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

}
```

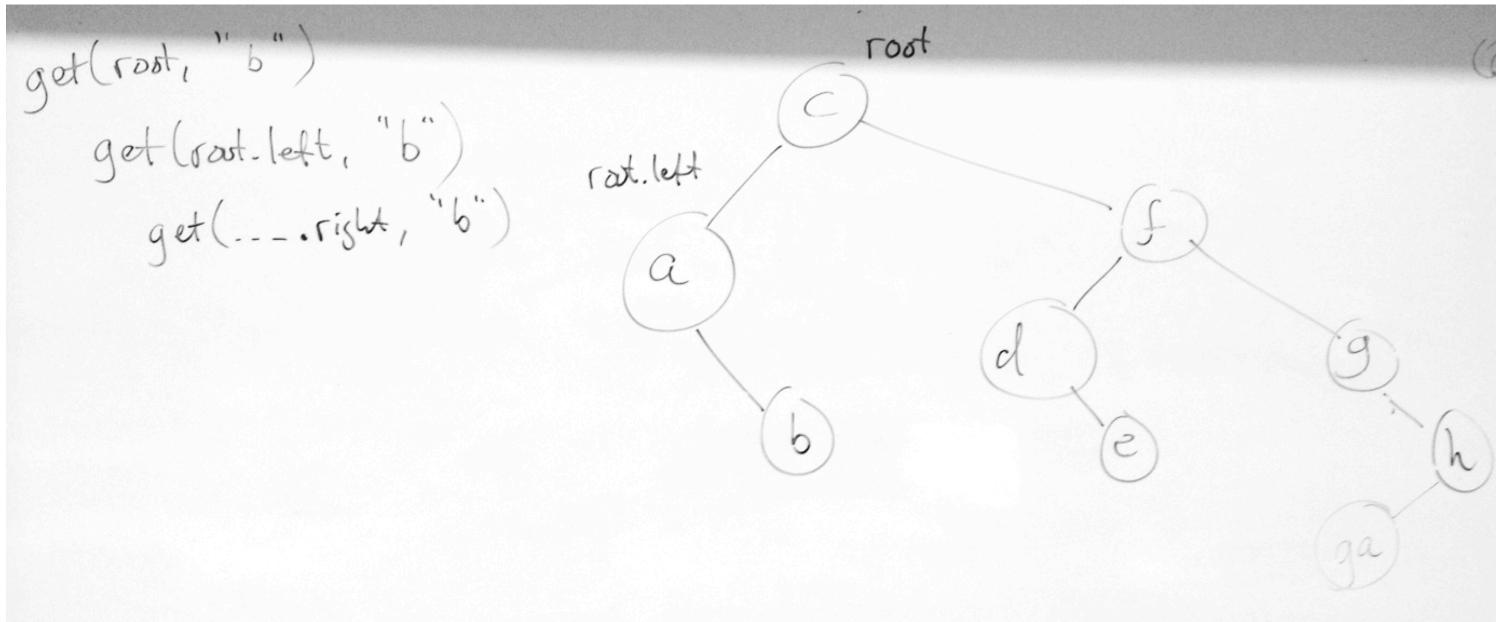
BST get

Get: skilar gildinu `val` fyrir lykilinn `key` eða `null` ef hann finnst ekki




```
public Value get(Key key) {
   return get(root, key);
}

private Value get(Node x, Key key) {
   if (x == null) return null;
   int cmp = key.compareTo(x.key);
   if      (cmp < 0) return get(x.left,  key);
   else if (cmp > 0) return get(x.right, key);
   else              return x.val;
}

public boolean contains(Key key) {
   return (get(key) != null);
}
```

get(Node x, Key key) hjálparfallið þegar leitað er að “b”



BST put

Put: látum lykilinn `key` fá gildið `val`

- leita fyrst, setja svo inn í tré
- stuttur, en lúmskur, endurkvæmur kóði

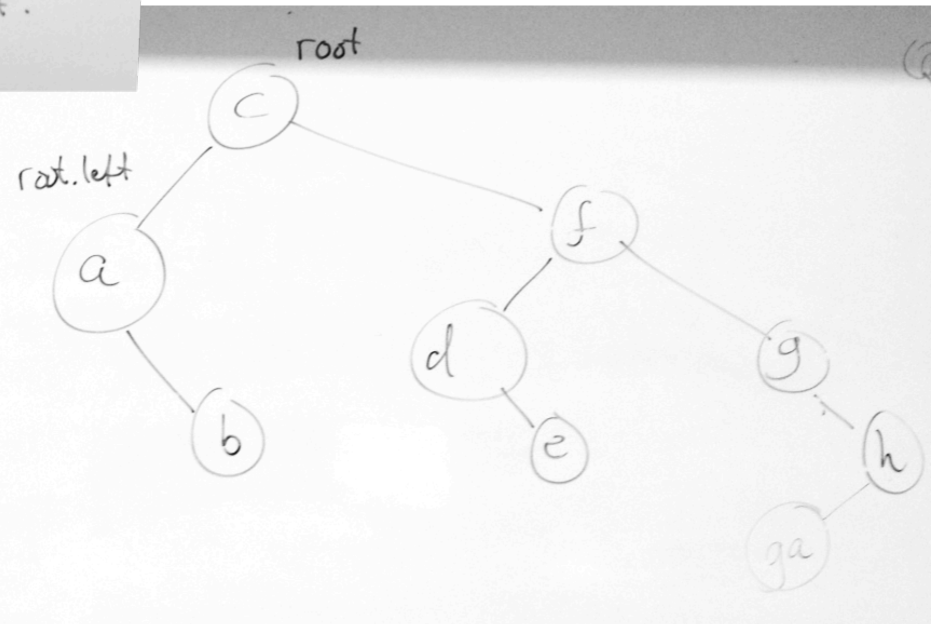
```
public void put(Key key, Value val) {
   root = put(root, key, val);
}

private Node put(Node x, Key key, Value val) {
   if (x == null) return new Node(key, val);
   int cmp = key.compareTo(x.key);
   if      (cmp < 0) x.left  = put(x.left,  key, val);
   else if (cmp > 0) x.right = put(x.right, key, val);
   else x.val = val;
   return x;
}
```

Skrifum yfir gamla gildið

```
put("ga")            Skilagildi → rótin í nýja trénu.
  put(root, "ga")
    put(f, "ga")
      put(g, "ga")
        put(h, "ga")
          x.left = put(null, "ga")
```

x er h-hnúturinn (arrow to x)

skilar nýjum Node hlut. (brace under put(null, "ga"))



BST mælingar

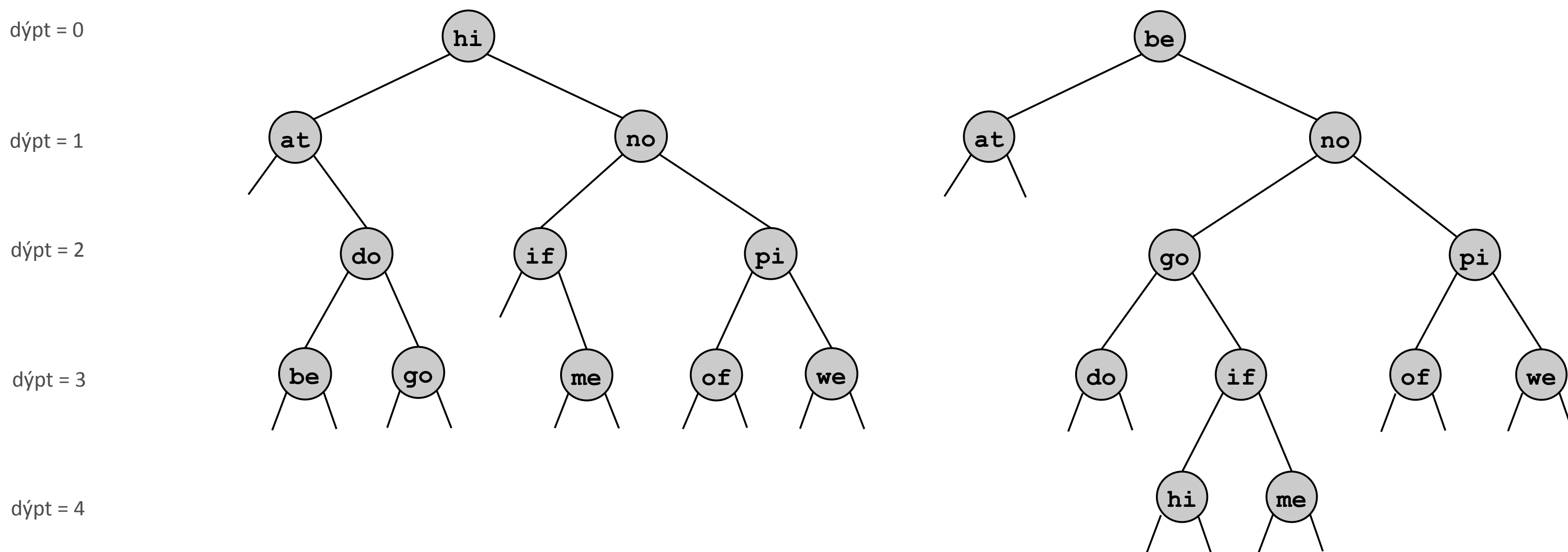
Niðurstaða: áberandi munur í hraða á einföldum lausnum og BST lausn

implementation	get	put	Moby	100K	200K	1M
unordered array	N	N	170 sec	4.1 hr	-	-
ordered array	$\log N$	N	5.8 sec	5.8 min	15 min	2.1 hr
BST	?	?	.95 sec	7.1 sec	14 sec	69 sec

Tími fyrir get/put

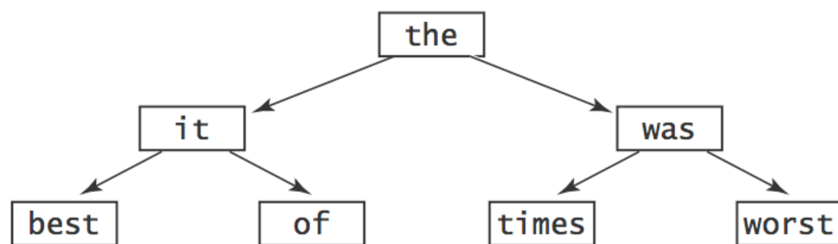
- Mörg mismunandi tré geta táknað sömu töflu
- Tíminn er í réttu hlutfalli við dýptina á hnútnum sem er með lykilinn

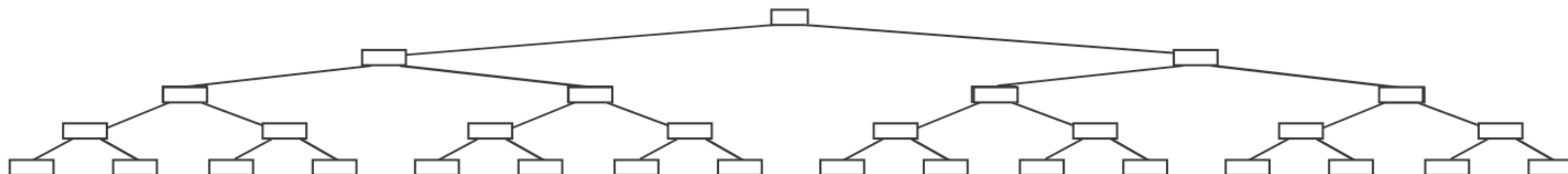
fjöldi hnúta upp að rót



BST tímakeyrsla

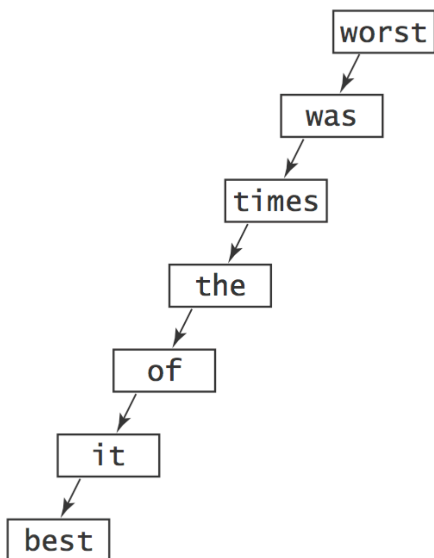
Besta tilfelli: Tréð er jafndjúpt, dýptin er í mesta lagi $\log_2(N)$

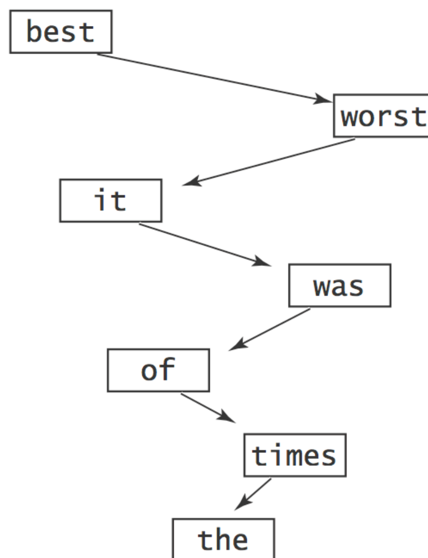


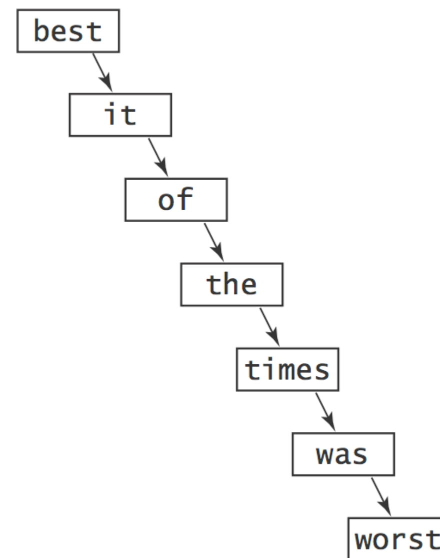


BST tímakeyrsla

Versta tilfelli: ef tréð er ójafnt, dýptin getur verið N

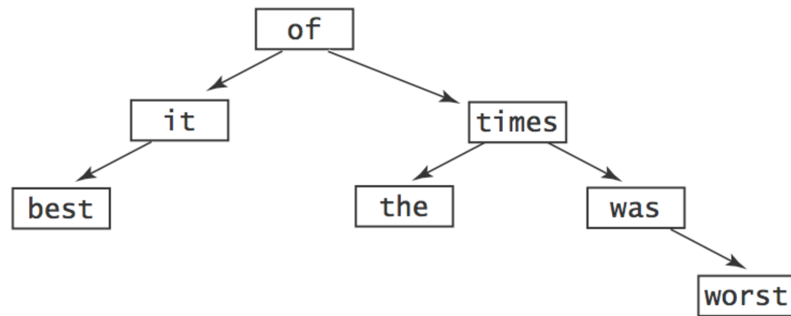


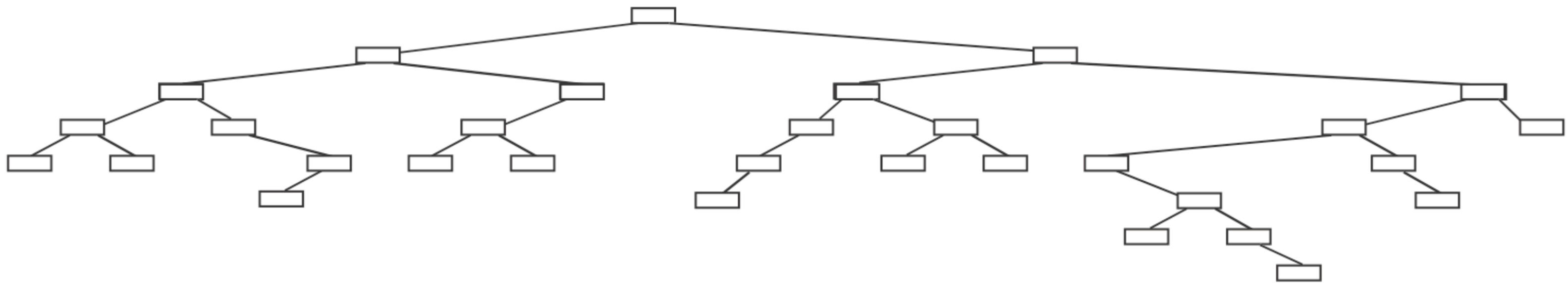




BST útfærsla

Meðaltals innsetningartími er $2\log_2 N$ ef lyklarnir birtast í handahófskenndri röð





Typical BSTs constructed from randomly ordered keys

BST útfærsla

- Það er hægt að tryggja O(log N) innsetningartíma fyrir tvíleitartré

- Trénu er breytt þegar nýr lykill bætist við til að tryggja að það sé flatt

- Erfitt að forrita rétt, TreeMap í java.util notar Red-Black trees til að gera þetta

Hakkatöflur

Hakkatöflur (hashtable) nota hakkafall (hash function) til að flýta fyrir leit.

Hakkafall, h(x): Key -> int, varpar lyklum í heiltölur

x.hashCode() í Java.

Hugmynd:

- Geymum lykla og gildi í tveimur fylkjum
- reiknum i = h(x) % M, M stærð fylkisins
- x á (helst) að vera geymt í sæti i
- auðvelt að leita
- auðvelt að setja inn

Hakkatöflur

Vandamál, tvö stök x1, x2 gætu bæði átt að vera í sama sæti, þ.e. h(x1) = h(x2) % M

Lausn:

- Ef x1 kom fyrst
- Þá getum við ekki sett x2 í sæti i
- Prófum sæti i+1,i+2,… %M þangað til við finnum laust sæti

Einfalt að leita: byrjum í sæti i og höldum áfram

Einfalt að setja inn: …

Útfærsla: Vikublað 13

Hakkaföll og hakkatöflur

Hakkafall reiknar heiltölu fyrir eitthvað inntak.

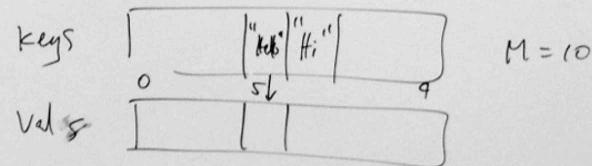
- allar mögulegar útkomur jafnlíklegar
- "slembið"
- hraðvirkt.
- sama gildið alltaf.

"Hello"

87, 100, 110, 110, 120

↓ ↓ ↓ ↓ ↓

\+ + + + =



lykilinn x, h(x)

h("Hello") = 305

h("Hi") = 405

java.util

Í java.util pakkanum er fullt af gagnagrindum

- Gagnagrind er klasi sem heldur utan um gögn á skipulegan hátt
- Array er eitt dæmi um einfalda gagnagrind
- Vector er aðeins betra dæmi
- Nokkur í viðbót sem við sjáum

Sama lýsing á java.util.Vector og við notuðum, miklu fleiri aðferðir. Til að nota þetta skrifum við

```
import java.util.Vector; // nær í pakkann
...
Vector<String> v = new Vector<String>();
```

Iterator

Eitt af því sem við gerum oft með vektora og aðrar gagnagrindur er að fara í gegnum þær með for lykkju.

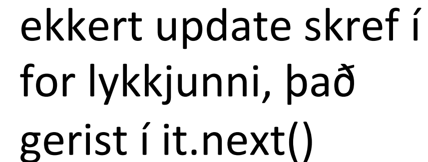
- Einfalt fyrir fylki og Vector, for(int i = 0, …)
- Aðrar gagnagrindur eru ekki í neinni “röð”
- Iterator skilin leysa þetta vandamál

```
public interface Iterator<E> {
    boolean hasNext(); // er eitthvað eftir?
    E next(); // náum í næsta stak
    void remove(); // yfirleitt ekki notað!
}
```

Iterator kemur í staðin fyrir int i í for lykkjunum.

Iterator dæmi

```
Vector<String> v = new Vector<>();
… // fyllum inn í v;

for(Iterator<String> it=v.iterator();
    it.hasNext(); ){
  String s = it.next();
  … // vinnum með s
}
```



Iterator dæmi

Þetta er svo algeng notkun að það er til sér útgáfa af for lykkjunni fyrir þetta.

```
for (E e : c) { … }
```

c þarf að hafa iterator() aðferð og útfærar Iterable<E> skilin (Vector gerir það) og E þarf að passa við Iterator<E> tagið á c

```
Vector<String> v = new Vector<>();
… // fyllum inn í v;

for(String s : v) { … }
```

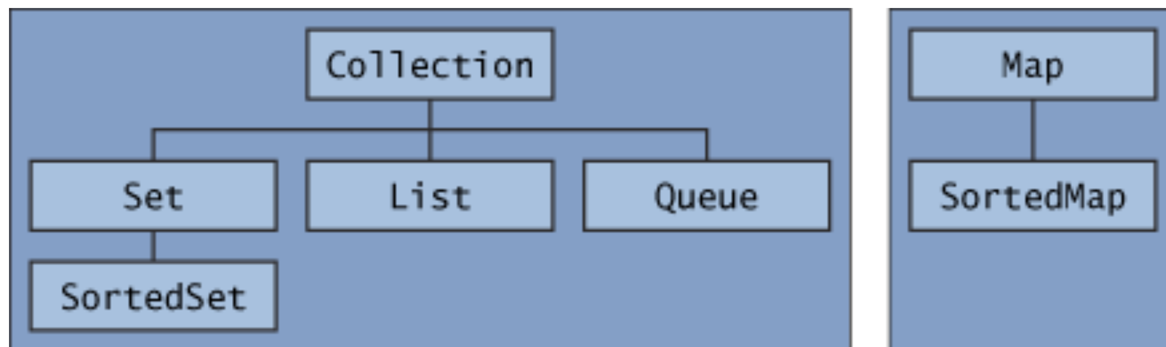
java.util

- 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

java.util

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));
```