Stockholms universitet

Programmering 2

Föreläsning 4: Datasamlingar – implementeringstekniker

Isak Samsten, VT22

Stockholms universitet

Datasamlingar i standardbiblioteket

- Generiska klasser: ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap
- Gränssnitt: Collections (alla), List (listor), Set (mängder) och Map (avbildningar)

Stockholms universitet

Gränssnitt: Collections

- Rotklass för samtliga samlingar
- Har metoder för att lägga till, ta bort och kontrollera existens av objekt
- Iterera över objekten i samlingen
- Returnera strömmar
- Returnera array av samlingen

Stockholms universitet

Gränssnitt: Collections

- Lägga till/ta bort element
  - `add(E e)`
  - `addAll(Collection<? extends E> c)`
  - `remove(E e)`
- Kontrollera om element finns
  - `contains(E e)`
  - `isEmpty()`
- Iterera (ex. med `for(E e : coll)`)
  - `Iterator<E> iterator()`
- Skapa array
  - `toArray(), toArray(E[] a)`

Stockholms universitet

Gränssnitt: List

- Supertyp för `ArrayList` och `LinkedList` (och andra listtyper)
- Ärver av `Collection`
- Har metoder för att hämta värden på en specifik position, ta bort värden på en specifik position och lägga till värden på en viss position
- Specifik iterator för listor

Stockholms universitet

Gränssnitt: List

- Lägga till/ta bort på position
  - `add(int i, E e)`
  - `remove(int i)`
- Hämta element på position
  - `get(int i)`
- Specifik iterator
  - `listIterator()`
    - Näst/föregående element eller index
    - Ta bort/lägg till/ändra element på index

Stockholms universitet

Gränssnitt: Set

- Supertyp för `HashSet` och `TreeSet` (och några andra klasser)
- Ärver av Collection
- Endast unika element
- Har metoder för att få delmängder, slå samman mängder och skillnaden mellan mängder – dvs. mängdoperationer

Stockholms universitet

Gränssnitt: Set

- Inga metoder utöver de som finns i Collection
- Markerar snarare samlingar som Set
- Konkreta implementationer däremot kan ha egna metoder, ex. första elementet eller alla element mellan två värden

Stockholms universitet

Gränssnitt: Map

- Sypertyp för avbildningar som HashSet och TreeSet
- Ärver inte av Collections
- Kopplar nycklar till värden
- Har metoder för att hämta ett värde/lägga till/ta bort värde givet en nyckel
- Iterera över nycklar, värden eller nyckel/värde-par

Stockholms universitet

Gränssnitt: Map

- Lägga till par:
  - `put(K k, V v)`
  - `putAll(Map<? extends K, ? extends V> o)`
- Ta bort par:
  - `remove(Object k)`
- Iterera:
  - `keySet()`
  - `values()`
  - `entrySet()`

Stockholms universitet

Datasamlingsoperationer

- Operationer på datasamlingar samlas i klassen `Collections`
  - `sort(List)`
  - `reverse(List)`
  - `shuffle(List)`
  - `max(Collection)`
  - `min(Collection)`

Stockholms universitet

Liknande för arrayer

- I klassen Arrays finns liknande metoder för arrayer
  - `sort(Object[])`
  - `sort(int[])`
  - `sort(double[])`

Stockholms universitet

Strömmar

- Man kan erhålla strömmar av samlingar med metoden `stream()`
- Strömmar är inte samlingar utan används för att aggregera, filtrera och ändra värden
- Håller inte element i minnet utan ”strömmar dem” från exempelvis en samling
- Eller från en fil; eller över nätverk

Samlingar i Java

Stockholms universitet

Konkret klass	Supertyp	Kännetecken
`ArrayList<E>`	`List<E>`	Lista, array-implementation, tillåter dubletter och null, har positionsoperationer t.ex. get(int index), add(int index, E ny), remove(int index)
`LinkedList<E>`	`List<E>`	Lista, länkad implementering, tillåter dubletter och null, har positionsoperationer
`HashSet<E>`	`Set<E>`	Mängd, hashtabell-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen osorterade
`TreeSet<E>`	`Set<E>`	Mängd, träd-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen sorterade, kräver att elementen skall kunna jämföras
`HashMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med hashtabell, håller elementen osorterade
`TreeMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med trädstruktur, håller elementen sorterade efter nyckelvärden, kräver att nyckelvärden skall kunna jämföras

Samlingar i Java

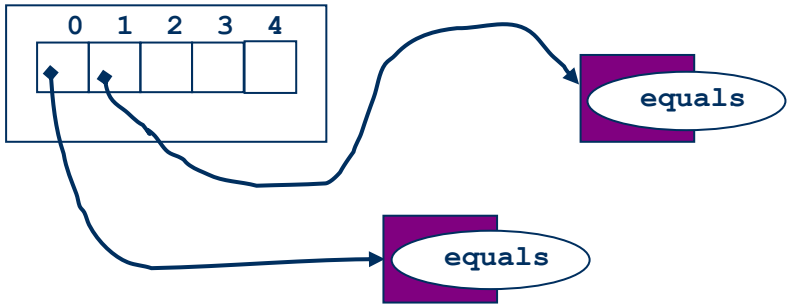
Stockholms universitet

Konkret klass	Supertyp	Kännetecken
`ArrayList<E>`	`List<E>`	Lista, array-implementation, tillåter dubletter och null, har positionsoperationer t.ex. get(int index), add(int index, E ny), remove(int index)
`LinkedList<E>`	`List<E>`	Lista, länkad implementering, tillåter dubletter och null, har positionsoperationer
`HashSet<E>`	`Set<E>`	Mängd, hashtabell-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen osorterade
`TreeSet<E>`	`Set<E>`	Mängd, träd-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen sorterade, kräver att elementen skall kunna jämföras
`HashMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med hashtabell, håller elementen osorterade
`TreeMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med trädstruktur, håller elementen sorterade efter nyckelvärden, kräver att nyckelvärden skall kunna jämföras

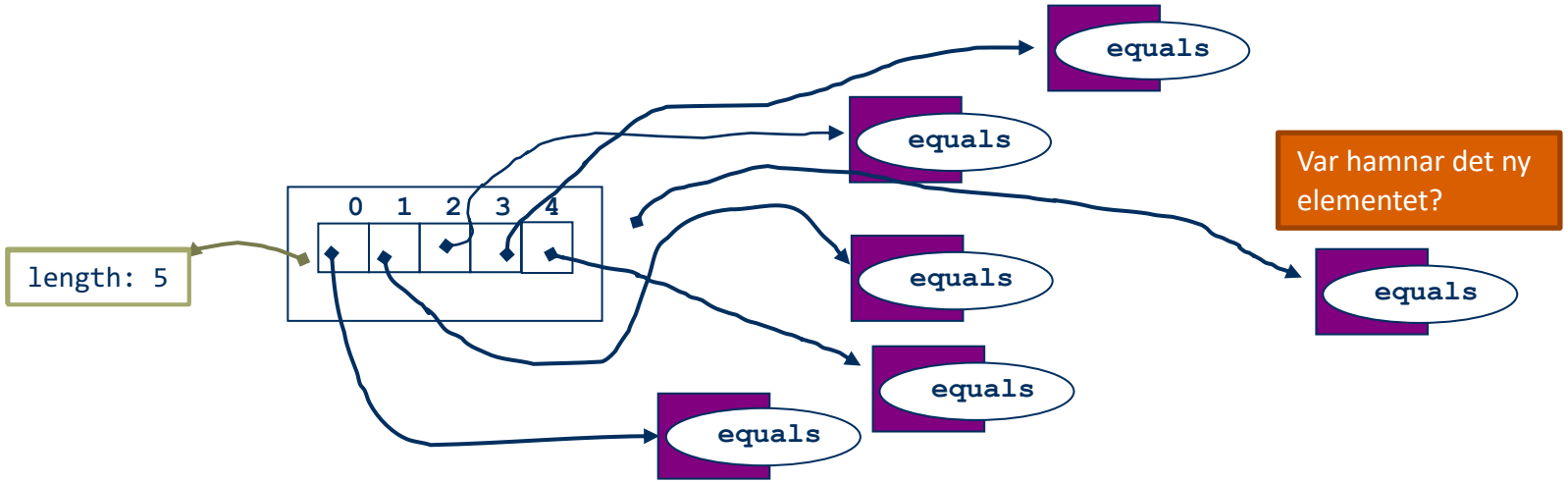
Utökningsbar array

Array (utökningsbar)

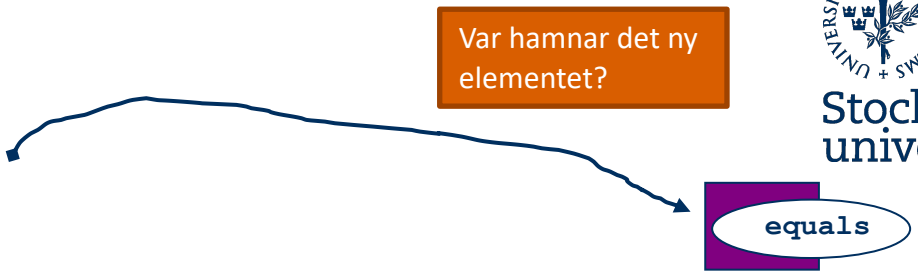
- Snabb indexering,
- Snabba tillägg/borttag i slutet
- Långsamma tillägg/borttag i början och mitten.
- Sökning genom iterering
- Kräver endast `equals` av objekten
- Används i klassen `ArrayList`



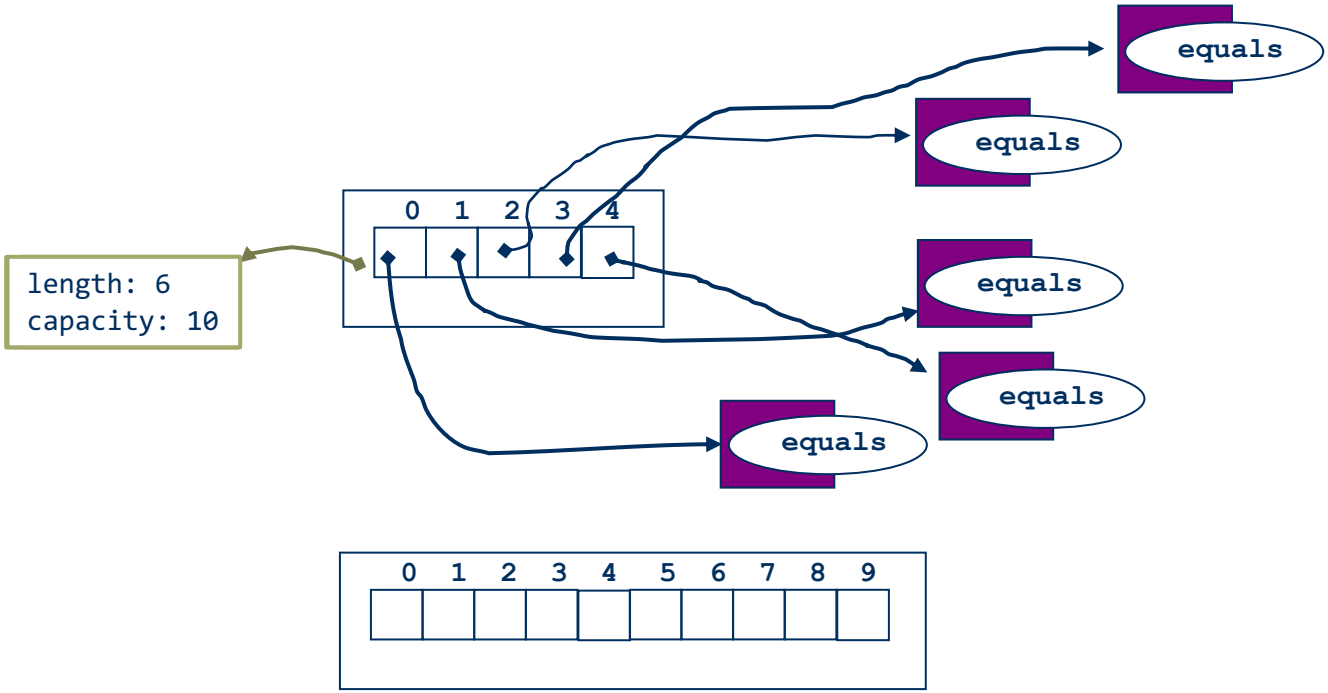
Lägga till element i utökningsbar array



- Snabba tillägg/borttag i slutet



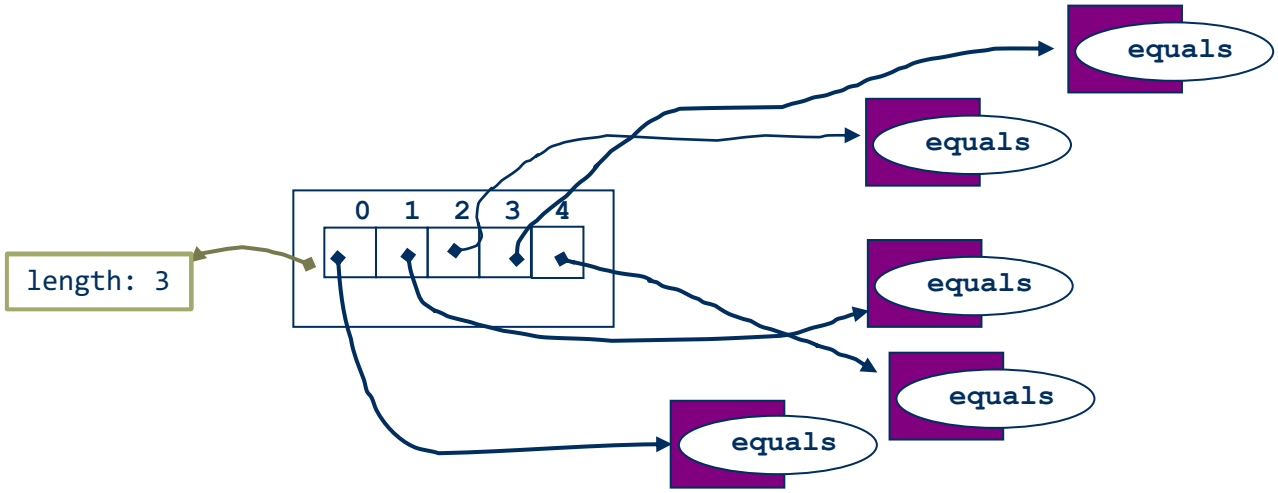
Lägga till element i utökningsbar array



- Ibland måste listan kopieras – viss kostnad

Stockholms universitet

Ta bort element ur utökningsbar array



- Långsamma tillägg/borttag i början och mitten.

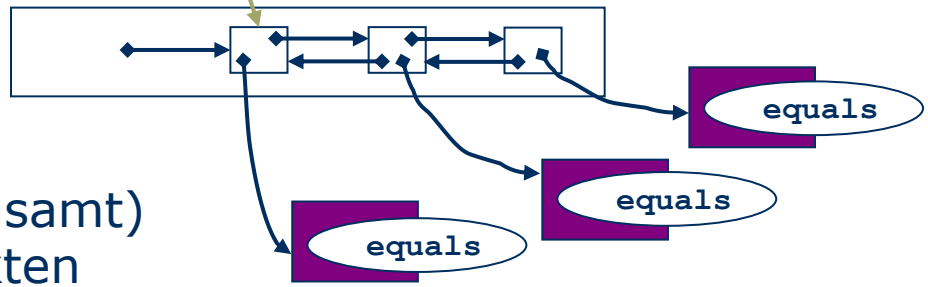
Samlingar i Java

Stockholms universitet

Konkret klass	Supertyp	Kännetecken
`ArrayList<E>`	`List<E>`	Lista, array-implementation, tillåter dubletter och null, har positionsoperationer t.ex. get(int index), add(int index, E ny), remove(int index)
`LinkedList<E>`	`List<E>`	Lista, länkad implementering, tillåter dubletter och null, har positionsoperationer
`HashSet<E>`	`Set<E>`	Mängd, hashtabell-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen osorterade
`TreeSet<E>`	`Set<E>`	Mängd, träd-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen sorterade, kräver att elementen skall kunna jämföras
`HashMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med hashtabell, håller elementen osorterade
`TreeMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med trädstruktur, håller elementen sorterade efter nyckelvärden, kräver att nyckelvärden skall kunna jämföras

Länkad lista

`Link`-objekt som pekar ut nästa och föregående `Link`-objekt. Varje `Link`-objekt innehåller också en referens till objektet vi sparar i listan



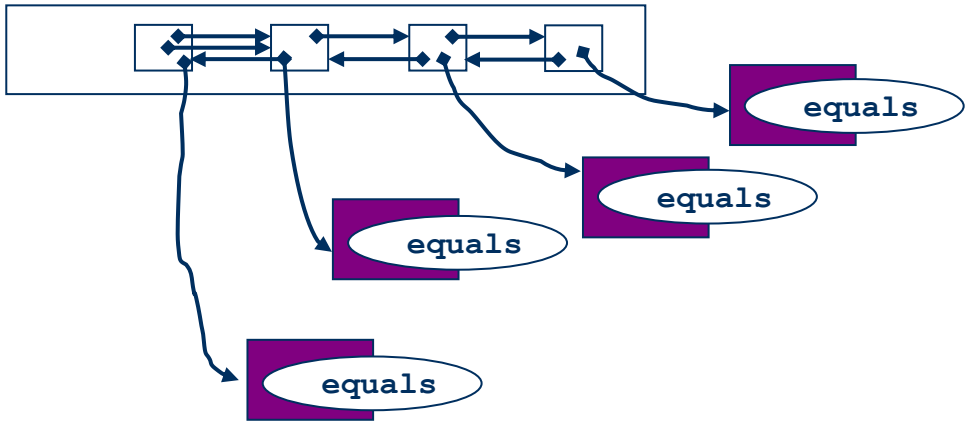
Länkad lista

- Långsam indexering
- Snabba tillägg/borttag
- Sökning genom iterering (långsamt)
- Kräver endast `equals` av objekten
- Används i klassen `LinkedList`

Stockholms universitet

Lägga till element i länkad lista

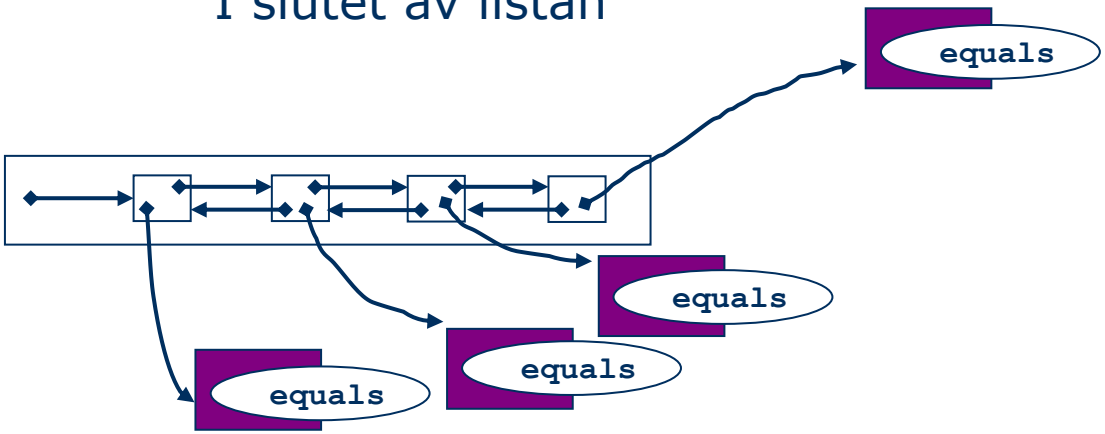
I början av listan



Stockholms universitet

Lägga till element i länkad lista

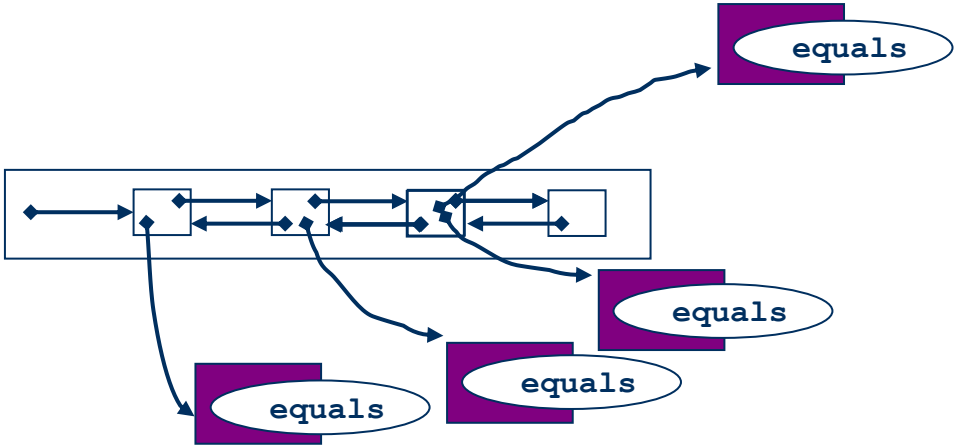
I slutet av listan



Stockholms universitet

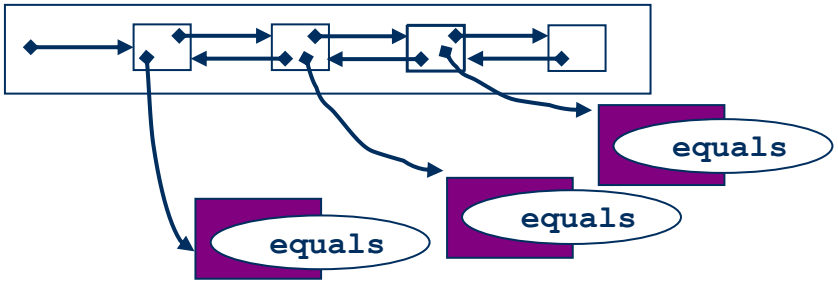
Lägga till element i länkad lista

I mitten på listan



Stockholms universitet

Ta bort element ur länkad lista



- Snabba tillägg/borttag

Stockholms universitet

Array-listor: egenskaper

- Sammanhängande minnesutrymme, befintlig array kan inte utökas eller minskas
- Ofta maximerad, kompletterad med en antalsvariabel
- Utökning kan simuleras genom allokering av större array och kopiering av gamla värden till den nya arrayen
- Direktåtkomststruktur: mycket snabb indexering
- Snabba tillägg och borttag i slutet (om extra utrymme finns)
- Långsamma tillägg/borttag i början/mitten (befintliga element måste flyttas)

Stockholms universitet

Länkade listor: egenskaper

- Dynamisk struktur: inga element från början, växer eller minskar efter behov
- Utrymme allokeras separat till varje nytt element
- Sekvensiell struktur: för att komma åt *i*-te elementet måste man börja från första och flytta fram *i* steg (långsam ”indexering”)
- Snabba tillägg i slutet om man har en extra referens till sista elementet
- Snabba tillägg och borttag i början (eller var som helst om man har en referens till det objekt som det nya objektet ska sättas in efter)
- Kräver extra utrymme för next/previous-pekarna i varje element

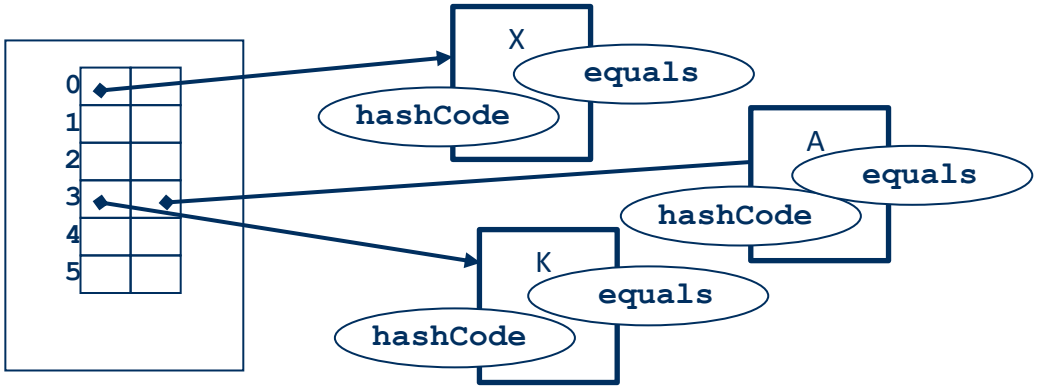
Samlingar i Java

Stockholms universitet

Konkret klass	Supertyp	Kännetecken
`ArrayList<E>`	`List<E>`	Lista, array-implementation, tillåter dubletter och null, har positionsoperationer t.ex. get(int index), add(int index, E ny), remove(int index)
`LinkedList<E>`	`List<E>`	Lista, länkad implementering, tillåter dubletter och null, har positionsoperationer
`HashSet<E>`	`Set<E>`	Mängd, hashtabell-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen osorterade
`TreeSet<E>`	`Set<E>`	Mängd, träd-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen sorterade, kräver att elementen skall kunna jämföras
`HashMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med hashtabell, håller elementen osorterade
`TreeMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med trädstruktur, håller elementen sorterade efter nyckelvärden, kräver att nyckelvärden skall kunna jämföras

Stockholms universitet

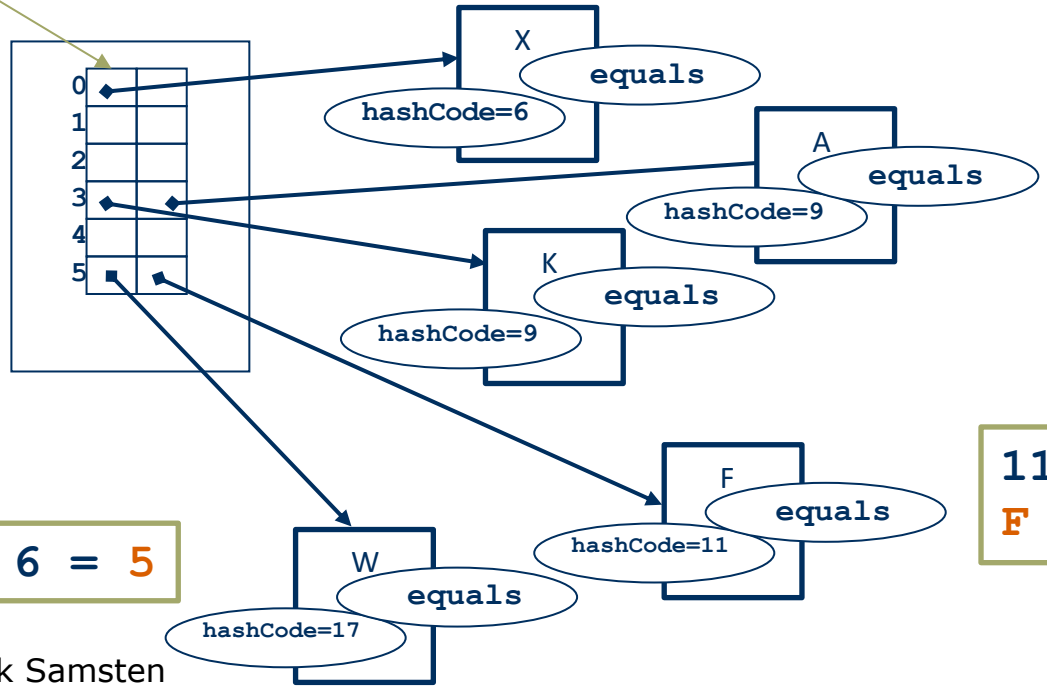
Hashmängder



- Använder `hashCode()` som index i en tabell (`hashCode() % capacity`)
- Använder även `equals()` om `hashCode()` krockar
- Mycket snabb sökning, tillägg och borttag.
- Oordnad.
- Används i `HashSet` och `HashMap`

Lägg till objekt i hashmängd

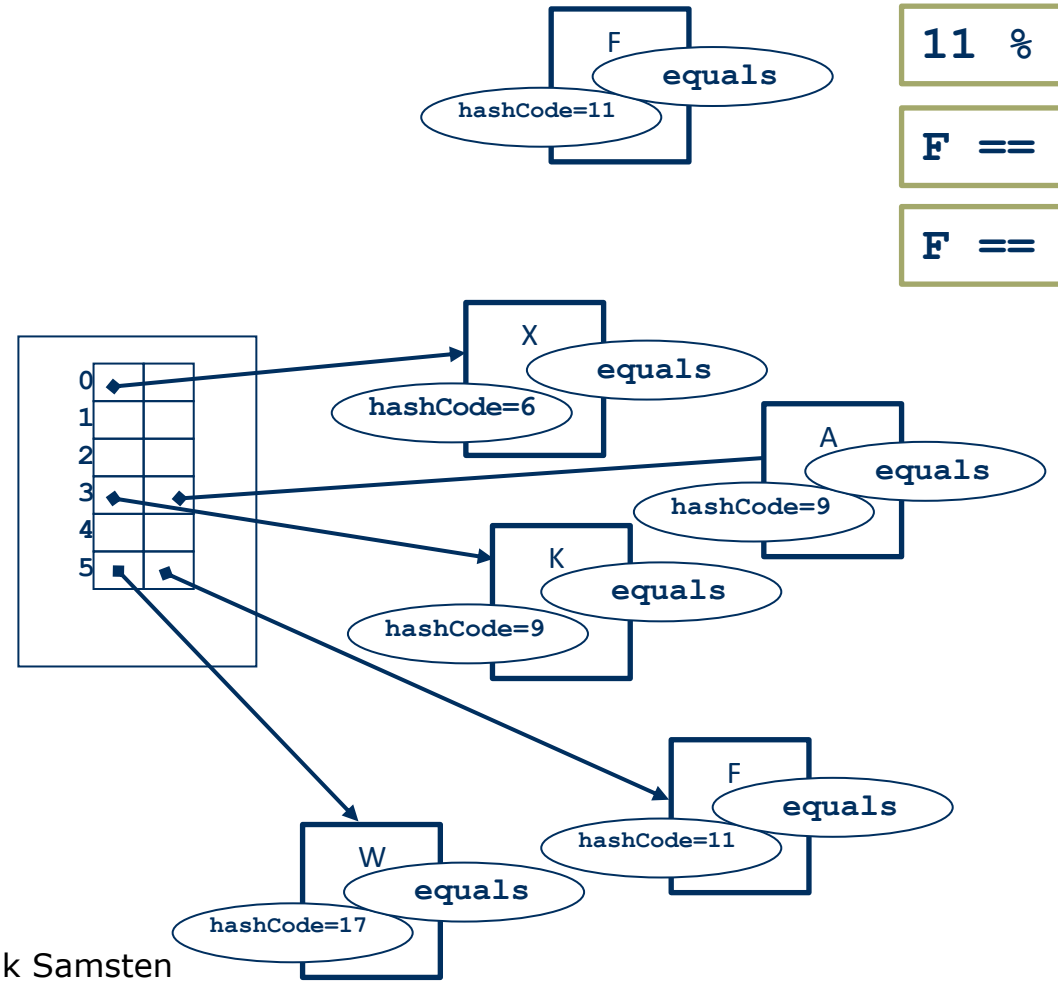
Varje plats i hasharrayen en länkad lista med element så vi kan hantera flera element med samma hash-code



`17 % 6 = 5`

`11 % 6 = 5`
`F != W`

Finns ett objekt i hashmängden?



Samlingar i Java

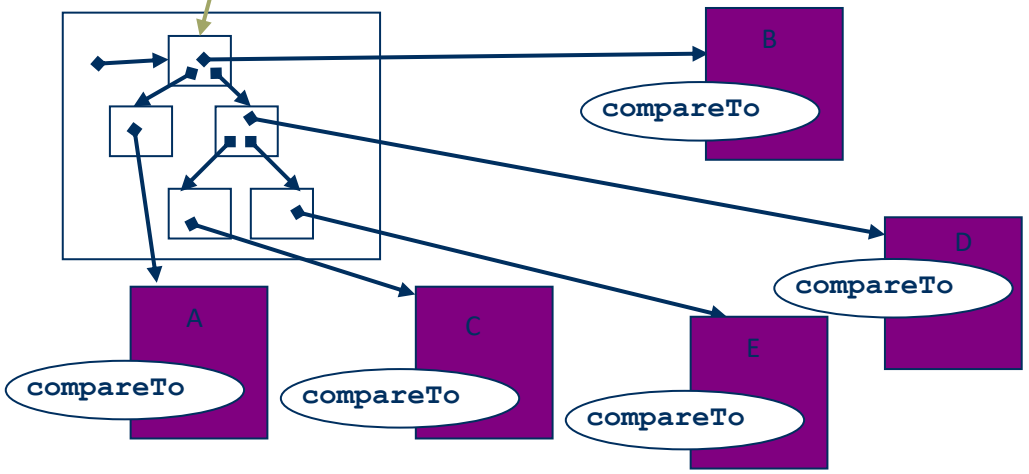
Stockholms universitet

Konkret klass	Supertyp	Kännetecken
`ArrayList<E>`	`List<E>`	Lista, array-implementation, tillåter dubletter och null, har positionsoperationer t.ex. get(int index), add(int index, E ny), remove(int index)
`LinkedList<E>`	`List<E>`	Lista, länkad implementering, tillåter dubletter och null, har positionsoperationer
`HashSet<E>`	`Set<E>`	Mängd, hashtabell-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen osorterade
`TreeSet<E>`	`Set<E>`	Mängd, träd-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen sorterade, kräver att elementen skall kunna jämföras
`HashMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med hashtabell, håller elementen osorterade
`TreeMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med trädstruktur, håller elementen sorterade efter nyckelvärden, kräver att nyckelvärden skall kunna jämföras

Stockholms universitet

`Node`-objekt som pekar ut mindre-än och större-än `Node`-objekt. Varje `Node`-objekt innehåller också en referens till objektet vi sparar i listan

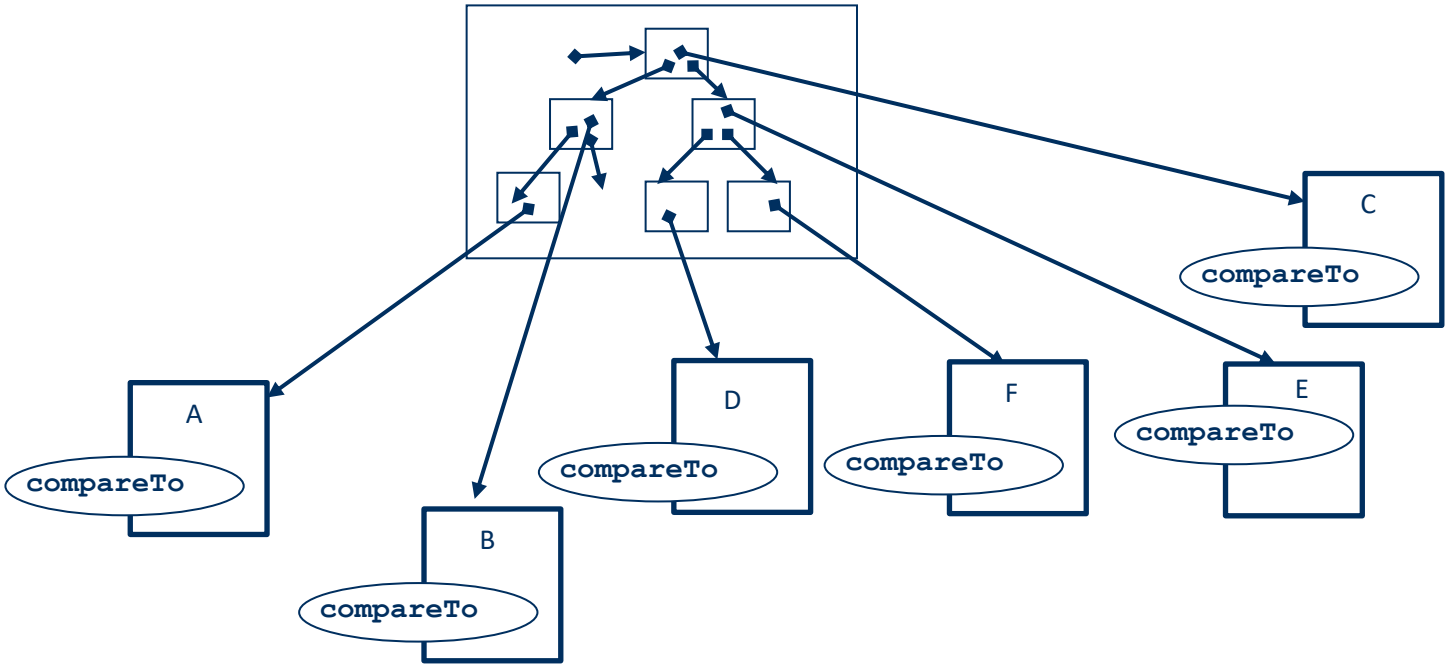
Trädmängder



- Ganska snabba sökningar, tillägg och borttag
- Objekten i sorteringsordning
- Måste kunna jämföras.
- Används i klassen `TreeSet` och i klassen `TreeMap`

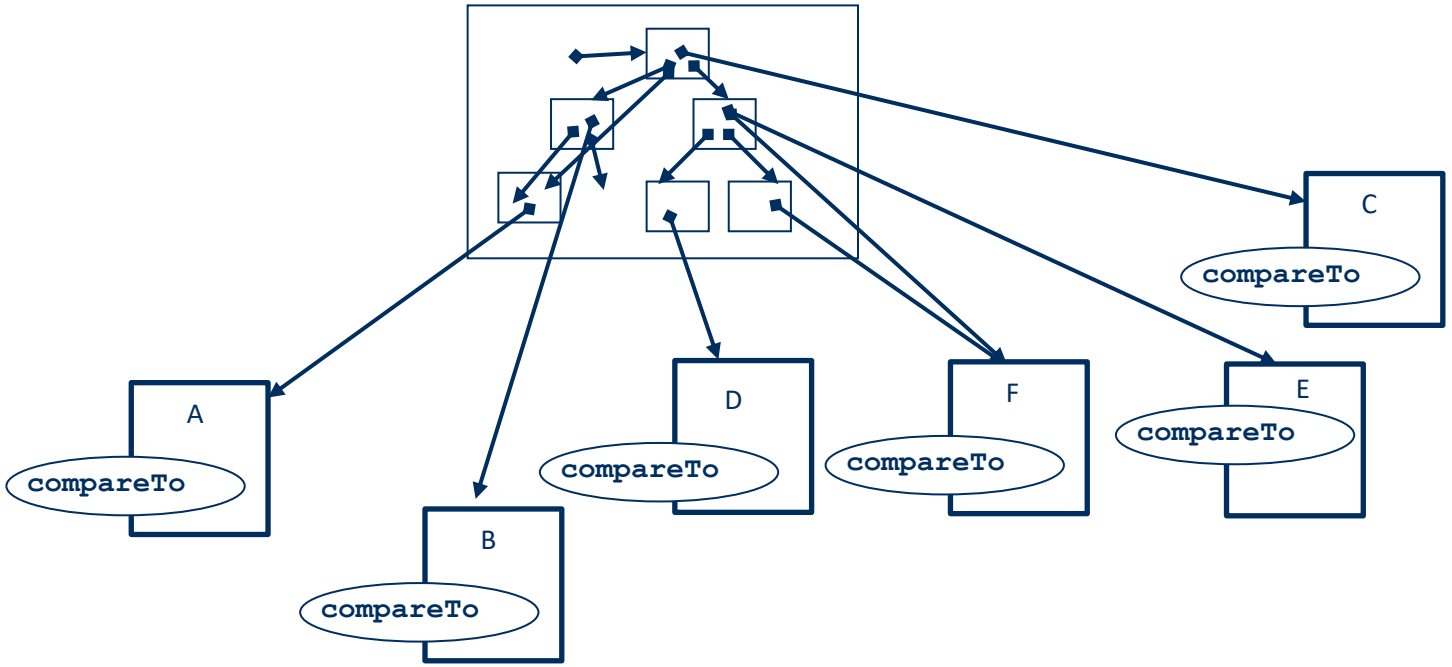
Stockholms universitet

Lägga till objekt i en trädmängd



Stockholms universitet

Ta bort objekt ur en trädmängd



Samlingar i Java

Stockholms universitet

Konkret klass	Supertyp	Kännetecken
`ArrayList<E>`	`List<E>`	Lista, array-implementation, tillåter dubletter och null, har positionsoperationer t.ex. get(int index), add(int index, E ny), remove(int index)
`LinkedList<E>`	`List<E>`	Lista, länkad implementering, tillåter dubletter och null, har positionsoperationer
`HashSet<E>`	`Set<E>`	Mängd, hashtabell-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen osorterade
`TreeSet<E>`	`Set<E>`	Mängd, träd-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen sorterade, kräver att elementen skall kunna jämföras
`HashMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med hashtabell, håller elementen osorterade
`TreeMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med trädstruktur, håller elementen sorterade efter nyckelvärden, kräver att nyckelvärden skall kunna jämföras

Stockholms universitet

Hashavbildning

- Kopplar nyckel av klassen K med värde av typen V
- Kräver att nyckel överskuggar `equals` och `hashCode`
- Mycket snabb på att hämta ett värde **givet** en nykel, så `map.get(nyckel)`
- Implementeras med hjälp av hash-tabeller

Samlingar i Java

Stockholms universitet

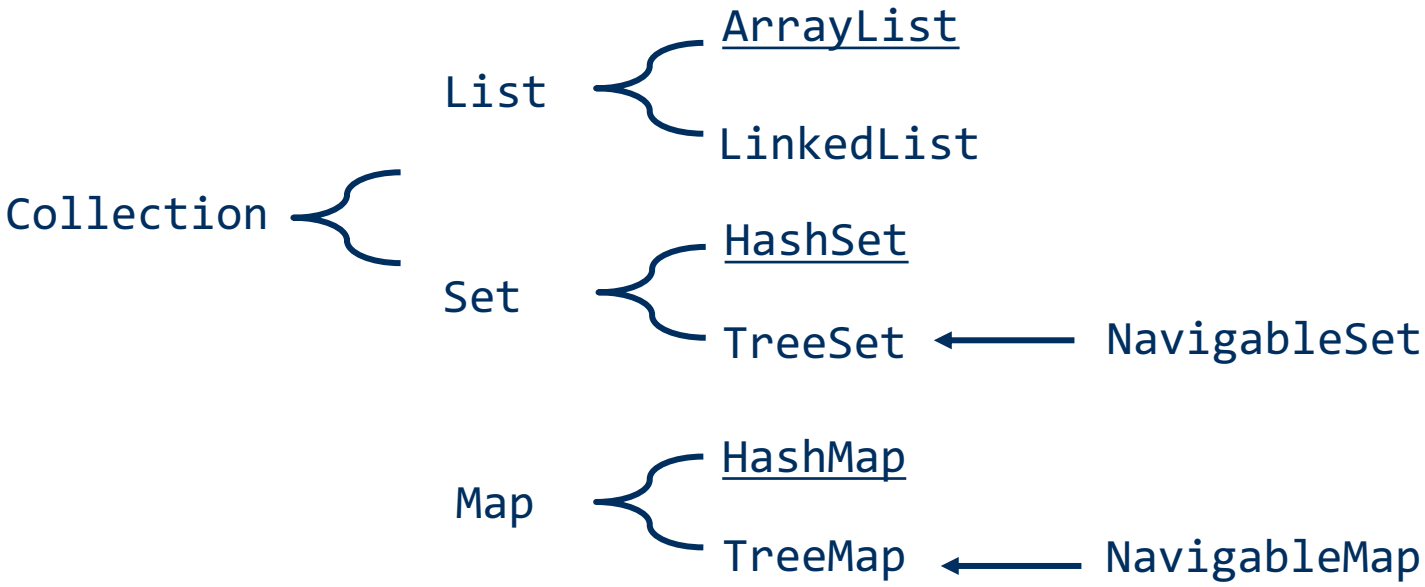
Konkret klass	Supertyp	Kännetecken
`ArrayList<E>`	`List<E>`	Lista, array-implementation, tillåter dubletter och null, har positionsoperationer t.ex. get(int index), add(int index, E ny), remove(int index)
`LinkedList<E>`	`List<E>`	Lista, länkad implementering, tillåter dubletter och null, har positionsoperationer
`HashSet<E>`	`Set<E>`	Mängd, hashtabell-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen osorterade
`TreeSet<E>`	`Set<E>`	Mängd, träd-implementering, filtrerar bort dubletter, har inga positionsoperationer, håller elementen sorterade, kräver att elementen skall kunna jämföras
`HashMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med hashtabell, håller elementen osorterade
`TreeMap<K, V>`	`Map<K, V>`	Avbildningstabell, identifierar elementen med nycklar, implementerad med trädstruktur, håller elementen sorterade efter nyckelvärden, kräver att nyckelvärden skall kunna jämföras

Stockholms universitet

Trädavbildning

- Kopplar nyckel av klassen K med värde av typen V
- Kräver att nyckel-klassen implementerar interfacet `Comparable<V>`
- Ganska snabb på att hämta värde givet en nyckel
- Bra eftersom nycklar är sorterade vid vissa tillämpningar
- Man kan exempelvis iterera nycklar i ordning
- Implementeras med hjälp av binära sökträd

`List`-klasser, `Set`-klasser och `Map`-klasser har i stort sett samma funktionalitet. Dessutom har `List`-klasser och `Set`-klasser liknande funktionalitet. Deras funktionalitet är därför deklarerad i olika gränssnitt.



Den understrukna klassen inom varje par rekommenderas om det inte finns speciell anledning att välja den andra.

Stockholms universitet

En liten notis om primitiva värden

- För varje primitivt värde finns en klass vars objekt kan representera dess värde
  - Kallas för *boxing* och *unboxing*

Privmitive type	Boxed reference typ
int	Integer
double	Double
long	Long
char	Character
boolean	Boolean

En liten notis om primitiva värden

Stockholms universitet

- Generiska parametrar kan **inte** användas för primitiva typer utan bara referenstyper
- Så:
  - ArrayList<int> fungerar **inte**
  - `ArrayList<Integer> a;` fungerar
- Pga. "boxing" och "unboxing" påverkas prestandan i vissa fall negativt

Skapara tusentals objekt som måste hanteras och konverteras

```
ArrayList<Integer> a = new ArrayList<>();
for (int i = 0; i < 1000000; i++) {
    a.add(i);
    int b = a.get(i);
    a.add(Integer.valueOf(i));
    int b = a.get(i).intValue();
}
```

Stockholms universitet

Går dock inte undkomma

- I prestandakritisk kod kan man använda specialklasser för primitiva värde
  - Finns dock inte i standardbiblioteket
- Ett annat alternativ är att använda arrayer (speciellt om man vet hur många värden man har)

Stockholms universitet

Komplexitetsklasser

- När vi pratar om att olika samlingar är olika snabba för olika operationer pratar vi om deras komplexitet
- Snabb: operationen utförs med konstant kostnad
- Ganska snabb: operationen utförs med logaritmisk kostnad
- Långsam: operationen utförs med linjär kostnad