

INF1000 – Forelesning 9

Hashmap

Eksempel: Flyreservasjon

HashMap

- Ofte har vi flere, mange objekter av en bestemt klasse - eks. :
 - elever på en skole
 - biler som har passert bomringen i Oslo
 - telefonsamtaler fra en bestemt person,.
- **HashMap** er en måte å lagre objekter der det er å raskt og enkelt finne igjen ett av objektene ut fra et kjennetegn
 - navnet til eleven,
 - registreringsnummeret til en bil, ...
- Et slikt kjennetegn som skiller ett objekt fra alle andre objekter, kaller vi en **nøkkel** (key)

Ulike versjoner i Java 1.4 (gammel) og Java 1.5/1.6 av HashMap

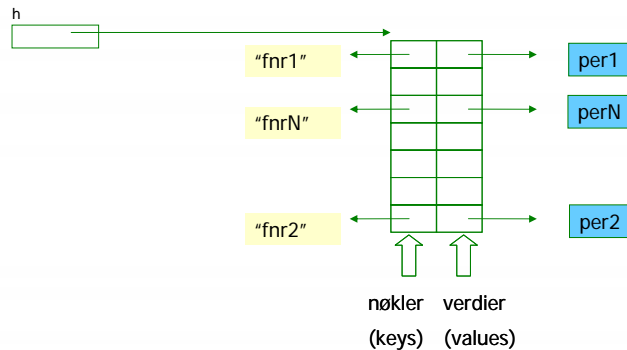
- Vi gjennomgår begge måtene, men anbefaler 1.5-måten
 - den hjelper deg mot visse feil (som ellers er lett å gjøre)
- 1.4 måten gjennomgås fordi mange gamle programmer inneholder slik kode

Objekter lagres med en søkenøkkel

- I en array legger vi inn objekter i en bestemt posisjon
 - Vi må vite om denne posisjonen når vi senere skal se på objektet
 - Posisjonen er en indeks mellom 0 og length-1
- I en HashMap oppgir vi en bestemt *nøkkel* når vi legger vi inn et nytt objekt (kalt *verdien*)
 - Vi oppgir denne nøkkelen når vi senere skal se på objektet
 - En HashMap 'vokser' når legger inn nye elementer
- Hashmap er mer fleksibel enn array!

Bilde av HashMap

```
HashMap<String,Person> h = new HashMap <String,Person>();
```



5

```
import java.util.*;
class BrukAvHashMap {
    public static void main (String[] args){
        HashMap<String,Person> h =
            new HashMap <String,Person>();

        String fnr1 = "30128812344";
        Person per1 = new Person(fnr1, "Harald Olsen");
        h.put(fnr1, per1);

        String fnr2 = "14109522547";
        Person per2 = new Person(fnr2, "Lena Torsen");
        h.put(fnr2, per2);

        Person p = h.get("30128812344");
    }
}

class Person {
    String fnr;
    String navn;

    Person(String fnr, String navn) {
        this.fnr = fnr;
        this.navn = navn;
    }

    String fåNavn() { return navn;}
}
```

Importer pakken java.util

Oppretter en HashMap og forteller hvilke klasser nøkkelen og verdier har.

Legg inn Person-objekt i HashMap'en

Legg inn Person-objekt i HashMap'en

Hent Person-objekt fra HashMap'en

Opprette en HashMap Java1.4 (gammel) og Java 1.5-1.6

- I starten av programmet:
`import java.util.*;`
Dette importerer pakken java.util hvor bl.a. klassen HashMap ligger.

- I klassen eller metoden som skal bruke HashMap'en **Java 1.4**:

```
HashMap h = new HashMap();
```

- I klassen eller metoden som skal bruke HashMap'en **Java 1.5** og nyere:

```
HashMap <String,Person> h = new HashMap <String,Person>();
```

Vi låser objektene til både nøkkelen og verdi-objektene til å være av disse typene.

NB: Hvis tabellen skal brukes av flere metoder i en klasse, deklarerer variabelen ovenfor i starten av klassen (som en objektvariabel).

Hvis tabellen kun skal brukes av en enkelt metode, er det naturlig å deklare HashMap variabelen ovenfor inni den aktuelle metoden.

7

Legge inn verdi-objekt i HashMap (samme i 1.4 og 1.5)

- Et hvilket som helst objekt kan legges inn som verdi i en HashMap
- I Java 1.5 må det være av den klassen vi har 'lovet' systemet.
- Når vi legger et verdi-objekt inn i HashMap'en, må vi samtidig oppgi et nøkkel-objekt av riktig type.
- Vi trenger denne nøkkelen når vi senere skal finne eller fjerne verdi-objektet i HashMap'en.

```
String fnr = "30126512345";
Person p = new Person(fnr, "Kari Olsen");
h.put(fnr, p);
```

Her lager vi et Person-objekt og legger det deretter inn i tabellen med fødselsnummeret som nøkkel.

8

- Dersom vi legger inn flere objekter med samme nøkkel, er det bare det sist innlagte objektet som blir liggende i tabellen (de andre overskrives):

```
Person p1 = new Person(...);
Person p2 = new Person(...);
Person p3 = new Person(...);
String navn = "Jens";
h.put(navn, p1); // p1 legges inn
h.put(navn, p2); // p2 legges inn og p1 overskrives
h.put(navn, p3); // p3 legges inn og p2 overskrives
```

- Noen ganger må vi konstruere en nøkkel ut fra flere variable for å få entydighet:

```
String lengdegrad = "67.3";
String breddegrad = "53.3";
String posisjon = lengdegrad + ";" + breddegrad;
Fjelltopp fjell = new Fjelltopp(posisjon, "Bjørnefjell");
h.put(posisjon, fjell);
```

Hente objekt fra HashMap – Java 1.4 og 1.5

Java 1.4: For å hente et objekt med utgangspunkt i nøkkelen:

```
// 1.4: Vi vil finne en person ut fra fnr:
Person p = (Person) h.get(fnr);
```

Vi må i starten skrive navnet på klassen som objektet tilhører i parentes - i dette tilfellet klassen Person.

Java 1.5: For å hente et objekt med utgangspunkt i nøkkelen, trenger vi ikke si hvilken klasse objektet har (det har vi jo sagt i deklarasjonen av HashMapen):

```
// 1.5: Vi vil finne en person ut fra fnr:
Person p = h.get(fnr)
```

Merk: å hente et objekt fra en HashMap slik som over medfører *ikke* at objektet fjernes fra HashMap'en (vi får bare en kopi av peker til objektet).

Fjerne objekt fra HashMap

- For å fjerne et objekt med gitt fødselsnummer som nøkkel:

```
h.remove(fnr);
```

- Dersom det ligger et objekt i HashMap'en med den gitte nøkkelen, blir objektet fjernet og setningen ovenfor returnerer med en peker til objektet som fjernes.
- Dersom det ikke ligger et objekt i HashMap'en med den gitte nøkkelen, returnerer setningen ovenfor verdien **null**.

Løp gjennom alle objekter i HashMap Java 1.4

- For å løpe gjennom alle objektene i en HashMap, lager vi en *oppramsing*:

```
Iterator it = h.values().iterator();
```

- Deretter kan vi se på hvert enkelt objekt i HashMap'en ved å gå i løkke:

```
while (it.hasNext()) {
    Person p = (Person) it.next();
    System.out.println("Navn: " + p.giNavn());
}
```

Løp gjennom alle objekter i HashMap Java 1.5

- Vi lager vi en oppramsing og låser samtidig det vi skal hente til en bestemt klasse:

```
Iterator<Person> it = h.values().iterator();
```

- Deretter kan vi se på hvert enkelt objekt i HashMap'en ved å gå i løkke:

```
while (it.hasNext()) {
    Person p = it.next();
    System.out.println("Navn: " + p.giNavn());
}
```

- Vi kan også i 1.5 benytte en *for-løkke* som automatisk lager en iterator:

```
for (Person p: h.values()) {
    System.out.println("Navn: " + p.giNavn());
}
```

13

To måter å løpe gjennom en HashMap – 1.5

- Løpe gjennom objektene (som på forrige foil):

```
Iterator<Person> it = h.values().iterator();
while (it.hasNext()) {
    Person p = it.next();
    <gjør noe med objektet p>
}
```

- Løpe gjennom nøklene:

```
Iterator <String> it = h.keySet().iterator();
while (it.hasNext()) {
    String nøkkel = it.next();
    <gjør noe med nøkkelen>
}
```

14

Metoder i HashMap

Metode	Eksempel	Beskrivelse
put	<code>h.put(nøkkel, objekt);</code>	Legg inn objekt med gitt nøkkel
get -1.4 get -1.5	<code>Person p = (Person) h.get(nøkkel);</code> <code>Person p = h.get(nøkkel);</code>	Finn objekt
remove	<code>h.remove(nøkkel);</code>	Fjern objekt
containsKey	<code>if (h.containsKey(nøkkel)) {</code> <code> // gjør et eller annet</code> <code>}</code>	Sjekk om nøkkel finnes i tabell
values	<code>Iterator it = h.values().iterator();</code>	Lag oppramsing av objektene
keySet	<code>Iterator it = h.keySet().iterator();</code>	Lag oppramsing av nøklene

15

Iterator (oppramsing)

	Eksempel	Beskrivelse deklarasjon
	<code>Iterator it1 = h.values().iterator();</code> <code>Iterator it2 = h.keySet().iterator();</code>	
hasNext()	<code>while (it.hasNext()) {</code> <code> < les neste og gjør noe;></code> <code>}</code>	returnerer true hvis flere objekter igjen i oppramsingen
next() 1.5 next() 1.4	<code>Person p = it.next();</code> <code>Person p = (Person) it.next();</code>	Finn neste objekt
remove()	<code>Person p = it.next();</code> <code>if (p.navn.equals("Arne"))</code> <code> it.remove();</code>	Fjern siste objekt som ble returnert med next()
	<code>while (it1.hasNext()) {</code> <code> Person p1 = it1.next();</code> <code>}</code> <code>for (Person p2 : h.values()){</code> <code> </code> <code>}</code>	To måter å gå gjennom alle verdiene (objektene) i h

16

```
import java.util.*;
import easyIO.*;
```

```
class Hasheksempel {
    public static void main(String[] argv) {
        In tastatur = new In();
        HashMap <String,Person> personregister = new HashMap <String,Person>();
        System.out.print("Antall personer som registreres : ");
        int ant = tastatur.inInt();

        for (int i = 0; i < ant; i++) {
            System.out.println("Gi neste person");
            Person p = new Person(tastatur);
            personregister.put(p.telefonnr, p);
        }
        // Skriv ut alle personobjektene
        System.out.println("Viser alle personer" +
            "(ukjent rekkefølge):");

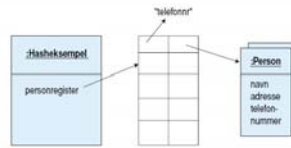
        for (Person p: personregister.values()){
            p.skrivData();
        }
    }
}
```

Eksempel fra boka
s.186

```
class Person {
    String navn, adresse, telefonnr;

    Person (In tastatur) {
        System.out.print("Oppgi navn : ");
        navn = tastatur.inLine();
        System.out.print("Oppgi adresse : ");
        adresse = tastatur.inLine();
        System.out.print("Oppgi telefonnummer : ");
        telefonnr = tastatur.inLine();
    }

    void skrivData() {
        System.out.println("Navn : " + navn);
        System.out.println("Adresse : " + adresse);
        System.out.println("Telefonnummer : " +
            +telefonnr);
    }
}
```



Klasser
Egenskaper
Prosedyrer

Flyreservasjon

- **Systemet** skal holde orden på alle selskapets flyvninger og reserverte seter på flyene
- En **flyvning** har en **kode**, et **avreisested** og en **destinasjon**, i tillegg til et **fly**, som har et **identifikasjonsnummer**
- Et fly består av **seterader**, med **seter**
- Systemet skal lese inn beskrivelse av flyene, med antall seter, **klasser** på de forskjellige seteradene, osv
- Det skal kunne **reservere seter**, **avbestille** og **skrive ut en oversikt** over flyets seter, med klasse og om det er ledig eller ikke

Klasseinndeling

- class Systemet
 - Inneholder kun main-metoden
 - Lager objekt av klassen under og kaller på brukerdiallog-metode.
- class Flyreservasjon
 - Inneholder brukerdiallogen og andre metoder + HashMap-tabeller for å holde orden på flyvningene.
- class Fly
 - Hvert objekt inneholder info om en flyet + alle seteradene og setene
- class Seterad
 - Setene i raden
- class Sete
 - Klasse og om det er opptatt eller ikke

class Systemet

```
import easyIO.*;
import java.util.*;

class Systemet {
    public static void main (String[] args) {
        String s1 = "Fly.txt";
        String s2 = "Bestillinger.txt";
        Flyreservasjon f = new Flyreservasjon(s1,
            s2);

        f.brukerdiallog();
    }
}
```

class Flyreservasjon

```
class Flyreservasjon {
    HashMap fly = new HashMap();
    HashMap<String, Flyvning> flyvninger
        = new HashMap<String, Flyvning>();

    Flyreservasjon(String s1, String s2) {
        lesFly(s1);
        lesReservasjoner(s2);
    }
    void lesFly(String filnavn) {...}
    void lesReservasjoner(String filnavn) {...}
    void brukerdialog() {...}
    ...
}
```

Datastruktur

Konstruktør som gjør initialisering (her: lese data fra fil)

Metoder for å lese fra fil og for å lese inn kommando fra bruker

Her kommer det metoder som skal kalles fra brukerdialogen

Flyreservasjon: brukerdialogen

- For hver kommando skal brukerdialogen kalle på en metode i klassen Flyreservasjon
- Sørg for å deklare alle de metodene som du kaller på fra brukerdialog-metoden
- Du kan vente med å fylle inn innholdet i disse metodene
- Eksempel: kaller brukerdialogen på metoden visFlyvning(), kan du skrive en "dummy-metode":

```
void visFlyvning() {
    System.out.println("Metoden visFlyvning utført");
}
```

Skrive ut flyvning

- Lag kode for metodene som kalles fra brukerdialogen
- Eksempel (i klassen Flyreservasjon):

```
void visFlyvning() {
    System.out.println("Flyvning: ");
    String flightKode = tast.inLine();

    Flyvning flight = <finn flyvingen ved oppslag i flyvninger>;

    flight.skrivUt();
}
```

Oppdraget delegeres videre til en metode i Flyvning-objektet som er aktuelt.

class Flyvning

Skriver ut litt informasjon om flyvningen og delegerer så ansvaret for utskrift av oppsettet i flyet til klasen fly.

```
class Flyvning {
    String flightkode;
    String avreisested;
    String destinasjon;
    Fly fly;
    void skrivUt() {
        System.out.println("Flight: " + flightkode);
        System.out.println("Fra: " + avreisested);
        System.out.println("Til: " + destinasjon);
        fly.skrivUt();
    }
}
```

Oppdraget delegeres videre til en metode i Fly-objektet

class Fly

- Skriver ut informasjon om flyet
- Delegerer videre til seteradene
- Delegerer videre til setene.

```
class Fly {
    String flykode;
    Seterad[] seterader;
    int skrivUt() {
        System.out.println("Flykode: " + flykode);
        for(int i=0; i<seterader.length; i++){
            seterader[i].skrivUt();
        }
    }
}
```

Og Fly delegerer videre

```
class Flyreservasjon {
    void brukerdiallog() {
        ...
        visFlyvning();
        ...
    }
    void visFlyvning() {
        ...
        flight.skrivUt();
        ...
    }
}
```

Vi har *ett* objekt av denne.

```
class Flyvning {
    skrivUt() {
        ...
        fly.skrivUt();
    }
}
```

Vi har *flere* objekter av disse.

```
class Fly{
    skrivUt() {...}
}
```

