INF1000

19. april 2010

Tips til oblig 4 + repetisjon

Christian Mahesh Hansen

Tips til

OBLIG 4

Oblig 4: Filmregister

- Kommandostyrt system som behandler informasjon om filmer
- Ved programstart: lese inn to datafiler, filmdata.txt og persondata.txt gitt i oppgaveteksten
- Skrive meny og lese inn valg fra brukeren
- Behøver *ikke* skrive informasjon til fil eller endre datafilene!

Filformat – filmdata.txt

- Én film pr. linje
- Hvert felt adskilt med tabulator-tegn (”\t”)

kode	tittel	år	regissør	skuespillere;…	sjangre	[*evt.tillegg*]
ACC3	The Accompanist	1992	cla4	ric12;yel1;rom2	a	
ACC11	Accumulator 1	1994	jan10	pet14;edi1;zde1	-	
ACC7	The Accused	1949	wil11	lor2;rob22;wen2;	n	
ACC5	The Accused	1988	jon6	kel2;jod1	2	
ACE1	Ace Drummond	1936	for2	joh29	a	
ACE6	Aces Go Places 4	1986	rin1	sam5;kar4	-	s=Aces Go Place

- Hvis et felt mangler data er dette angitt med -

Feltene i filmdata.txt

1. **kode** er en unik kode for hver film, og består av tre store bokstaver etterfulgt av et tall (som går fra 1 og oppover)
2. **tittel** er navn på filmen
3. **år** er året den kom
4. **regissør** er en kode som angir regissøren til filmen, og består av tre små bokstaver etterfulgt av et tall (som går fra 1 og oppover)
5. **skuespillere** er en rekke med koder som angir skuespillere, disse angis på samme måte som regissør, men kan være flere og er da adskilt med semikolon (";")
6. **sjangre** er en *sammenhengende tekststreng* som angir sjangre for filmen, i form av enkelt-bokstav-koder og enkelt-sifre
7. [***evt.tillegg***]: For noen av filmene er det mer enn 6 felt i linjen, den eneste av disse vi skal bruke i obligen er den som heter s=Serienavn

Filformat – persondata.txt

- Én person pr. linje
- To felter adskilt med tabulator-tegn (”\t”)

kode	navn
aam2	Aamir Bashir
aar1	Aaron Yoo
aar10	Aaron Stanford
aar11	Aaron Spann
aar12	Aaron Ruell
aar13	Aaron Ashmore

- Feltet kode svarer til kodene for regissør og skuespillere i filmdata.txt
- Feltet navn er en tekststreng som inneholder personens fulle navn

Meny og kommandoer

Eksempler på kommandoer du kan taste inn:

.	= Vis statistikk
AAA1	= Vis info om en film
AAA	= Finn film
tom1	= Vis info om en person
tom	= Finn person
90	= Vis info om et tiår
2009	= Vis info om et år (*)
a	= Vis info om en sjanger (*)
?	= Vis meny
q	= Avslutt

Kommando ('?' = meny): _

Deloppgaver

- **Vis statistikk:** Skal skrive ut totalt antall filmer, og antall filmer i hvert av tiårene 1980-1989, 1990-1999, og 2000-2009.
- **Vis info om en film,** og **Finn film**: Skal la brukeren angi en film, og skriver deretter ut følgende informasjon om filmen: tittel, år, fullt navn til regissør, og fullt navn til skuespillerne. Brukeren skal kunne angi film på minst 2 forskjellige måter:
  - *kode:* Hvis det som brukeren tastet inn var koden til en av filmene (f.eks. "AVA1" for Avatar), skal info om den filmen skrives ut.
  - *søk:* Hvis bruker bare tastet inn tre store bokstaver, skal programmet skrive ut en liste med filmene hvor tittelen begynner med disse bokstavene, og brukeren skal kunne velge ønsket film fra listen. Du kan bruke de unike filmkodene i datafilen til dette, disse ser bort fra "The " og "A " i begynnelsen av filmnavn.
  - *navn* (*valgfri ekstra-oppgave*) (*): Hvis du ønsker det kan du også implementere andre måter å angi filmer på, f.eks. med full tittel til filmen eller full tittel etterfulgt av år. Hvis du vil kan du også vise enda mer informasjon om filmen, f.eks. sjangre eller evt. serie som filmen tilhører.
- **Vis info om en person,** og **Finn person:** Skal fungere omtrent som kommandoen ovenfor, med de samme 2 eller 3 måter å angi ønsket person på (men gjerne med små bokstaver i stedet). Informasjonen som vises skal inneholde filmene som personen regisserte, og de som hun spilte i.
- **Vis info om et tiår:** Hvis brukeren taster inn to tallsifre, og det siste er 0, skal programmet vise følgende to ting: (a) Regissøren som regisserte flest filmer det tiåret, og (b) Filmene som står i 2 eller flere av filmlistene 1-5 i det tiåret. Filmlistene er angitt med kodene 1-5 i felt nr. 6 for hver film, f.eks. kode "2" står på filmer som har vunnet Oscar.
- **Vis info om et år** (*)**:** Hvis brukeren taster inn et årstall mellom 1900 og 2010 så skal programmet vise følgende to ting om det året: (a) "Årets sjanger" blant comedy, fantasy, horror, eller science-fiction: her skal programmet finne hvilken av disse 4 sjangrene forekommer i flest filmer det året, basert på sjanger-kodene c, f, h, s. Skriv også ut antall filmer resultatet er baserert på; og (b) Hvilken film står i flest æreslister (1-5) det året (uten hensyn til sjanger).
- **Vis info om en sjanger** (*)**:** Taster bruker bare en av bokstavene a, E, H, x (som står for a=action, E=eventyrfilm, H=superhero, x=disaster), så skal programmet vise følgende to ting om valgt sjanger: (a) Skuespilleren som spilte i flest filmer i sjangeren; og (b) Navnene på filmseriene som har minst en film innen sjangeren.

Sjangerkoder/filmlister

- Sjangerkodene er:
  - a=action A=animation b=biographical c=comedy C=children D=documentary d=drama e=epic E=adventure f=fantasy g=computer_animation(datagrafikk) h=horror H=superhero k=crime m=musical n=film_noir r=romantic_comedy R=romance s=science_fiction S=sports t=thriller w=war W=western x=disaster
- Filmistene er:
  - 1=Står i listen Films considered the greatest ever fra Wikipedia
  - 2=Står i listen List of Academy Award-winning films (filmer som fikk Oscar)
  - 3=Står i listen Palme d'Or (filmer som vant i Cannes)
  - 4=Står i listen Sight & Sound (liste omtalt i link 1= ovenfor).
  - 5=Står i listen AFI's 100 Years... 100 Movies (10th Anniversary Edition)

Leveransen

- **Oblig4.java:**
  - Denne filen skal inneholde det ferdige Java-programmet ditt (all Java-kode skal ligge i én fil).
  - Før du leverer lag en kommentar øverst i Oblig4.java der du skriver noe om besvarelsen din generelt, f.eks. om du ikke klarte å bli ferdig med en deloppgave, eller at du fullførte alle deloppgavene.
  - Skriv også hvilken leveringsmåte du valgte for UML-klassediagrammet (se neste punkt).
- **UML-klassediagram:**
  - Du skal levere et UML-klassediagram over systemet ditt.
  - Diagrammet skal inneholde navnene på klassene, og de viktigste koblingene mellom disse, illustrert som forhold (linjer) mellom klassene.
  - Gi navn på forholdene, og angi antall på begge sider av disse.
  - Du skal velge én av følgende tre måter å levere diagrammet på (angi hvilken du velger):
    - på papir
    - elektronisk via Joly
    - via mail
  - For elektronisk levering bruk en av disse filtypene: .pdf, .png, .gif, .jpg, .txt.

Hint

- Skall-fil med klasser og noen metoder ligger ute, men forsøk å sette opp selv først! Lærerikt mhp. eksamen…
- **UML-klassediagram:** Se eksempel på side 236 i læreboka
- **String.contains(”tekst”):** teste om en String inneholder en annen String, returnerer boolean

Repetisjon

HASHMAP

Holde orden på objekter - HashMap

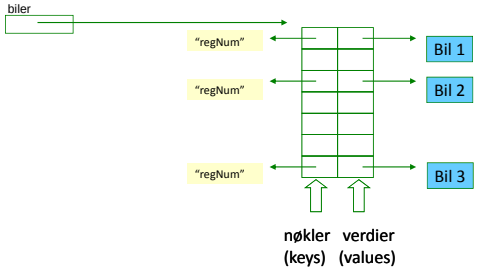
- Ofte har vi flere, mange objekter av en bestemt klasse - eks. :
  - Elever på en skole
  - Biler som har passert bomringen i Oslo
  - Telefonsamtaler fra en bestemt person
  - …
- Vi kan bruke arrayer, men da må vi passe på at vi har
  - Nok plass
  - Lete gjennom hele arrayen for å finne igjen objeker
- HashMap:
  - Dynamisk størrelse
  - Rask uthenting av objekter vha. nøkler (personnr., registreringsnummeret, etc.)

13

Hvordan vi tenker oss en HashMap

En HashMap er som en slags dobbelt-array

```
HashMap<String,Bil> biler =
       new HashMap<String,Bil>();
```



14

Metoder i HashMap

Metode	Eksempel	Beskrivelse
put	`h.put(nøkkel, objekt);`	Legg inn objekt med gitt nøkkel
get -1.4 get -1.5	`Person p = (Person) h.get(nøkkel);` `Person p = h.get(nøkkel);`	Finn objekt
remove	`h.remove(nøkkel);`	Fjern objekt

containsKey	`if (h.containsKey(nøkkel)) {`
	`  // gjør et eller annet`
	`}`
		Sjekk om nøkkel finnes i tabell

values	`Iterator it = h.values().iterator();`	Lag oppramsing av objektene
keySet	`Iterator it = h.keySet().iterator();`	Lag oppramsing av nøklene

15

Iterator (oppramsing)

	Eksempel
	`Iterator it1 = h.values().iterator();`
	`Iterator it2 = h.keySet().iterator();`
		Beskrivelse
		deklararasjon

hasNext()	`while (it.hasNext()) {`
	`  < les neste og gjør noe>;`
	`}`
		returnerer true hvis flere objekter igjen i oppramsingen

next() 1.5 next() 1.4	`Person p = it.next();` `Person p = (Person) it.next();`	Finn neste objekt

remove()	`Person p = it.next();`
	`if (p.navn.equals("Arne"))`
	`  it.remove();`
		Fjern siste objekt som ble returnert med next()

	`while (it1.hasNext()) {`
	`Person p1 = it1.next();`
	`}`
	`for (Person p2 : h.values()){`
	`  ....`
	`}`
		To måter å gå gjennom alle verdiene (objektene) i h

Repetisjon

KLASSER OG OBJEKTER

Klasser

- En klasse er en ”oppskrift” som forteller hvordan objekter skal se ut

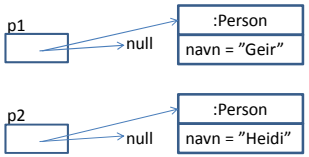
```
class Person {
    String navn;

    Person(String navn) {
        this.navn = navn;
    }
    String getNavn() {
        return navn;
    }
    void setNavn(String navn) {
        this.navn = navn;
    }
}
```

Objekter

- Et objekt er en *instans* av en klasse
- Lager nye objekter med **new**:

```
Person p1;
p1 = new Person(”Geir”);
Person p2;
p2 = new Person(”Heidi”);
```



Repetisjon

LESING FRA/SKRIVING TIL FIL MED EASYIO

Lese fra og skrive til fil

- Klassene In og Out i easyIO
  - Les dokumentasjonen!
- In og Out + Format
  - brukes i INF1000
  - Format brukes til mer 'finjustert' formattering
  - Det er flere metoder enn de som gjennomgås
- easyIO ble laget fordi javas innebygde IO-metoder var for kompliserte
  - java bedre nå
  - men fortsatt noe vanskeligere enn easyIO

21

Eksempel

Vi importerer pakken easyIO.

Vi åpner filen for lesing

Her leses hele første linje av filen

```
import easyIO.*;

class LesForsteLinje {
    public static void main (String[] args) {

        In fil = new In("filnavn");

        String s = fil.inLine();

        System.out.println("Første linje var: "
             + s);
    }
}
```

22

Lese ord for ord (item)

- Metoder:
  - inInt() for å lese et heltall
  - inDouble() for å lese et flyttall
  - inWord() for å lese et ord
  - lastItem() for å sjekke om slutten av filen er nådd
- Eksempel: lese en fil tall for tall

```
In fil = new In("item.txt");
while (!fil.lastItem()) {
    int k = fil.inInt();
    System.out.println("Tallet var " + k);
}
```

23

Lese ord for ord: skilletegn

Hopper over skilletegn mellom ordene man leser:

- linjeskift-tegnene (+ noen sære tegn) er alltid skilletegn
- Hvis man ikke gjør noe er også blanke, tab,... skilletegn
- Brukeren kan også spesifisere skilletegn:
  - String egneSkilletegn = "F(,)";
  - i = inInt(egneSkilletegn);
  - ***før*** det neste ord leses ignoreres nå kun linjeskift, samt tegnene i 'egneSkilletegn'

24

Lese tegn for tegn (ikke så mye brukt)

- inChar(false) returnerer det første uleste tegnet (uansett om det er skilletegn eller ikke)
  - Kan kopiere en fil tegn for tegn uansett hva den inneholder
- inChar(true) returnerer det første uleste tegnet som ikke er skilletegn
- Ser etter slutten med **endOfFile()**

25

Lese linje for linje

- Metoder:
  - readLine() for å lese en linje
  - inLine() for å lese resten av en linje (leser neste linje hvis det ikke er mer igjen enn linjeskift på nåværende linje)
  - endOfFile() for å sjekke om slutten av filen er nådd
- Eksempel: lese en fil linjevis

```
In fil = new In("fil.txt");
while (!fil.endOfFile()) {
    String s = fil.readLine();
    System.out.println(”Linjen var ” + s);
}
```

26

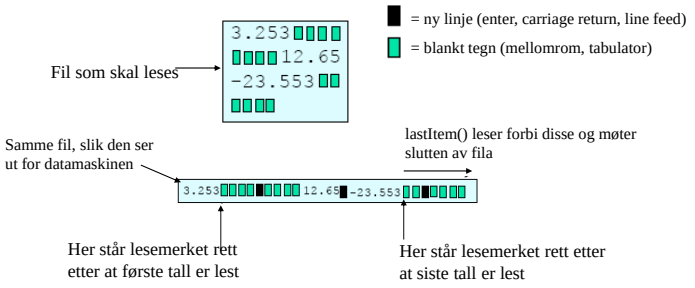
Program som leser en tekstfil linje for linje:

```
import easyIO.*;
class LinjeForLinje {
    public static void main (String[] args) {
        In innfil = new In(”filnavn");
        String[] s = new String[100];
        int ant = 0;
        while (!innfil.endOfFile()) {
            s[ant] = innfil.readLine();
            ant = ant + 1;
        }
        for (int i=0; i<ant; i++) {
            System.out.println(s[i]);
        }
    }
}
```

27

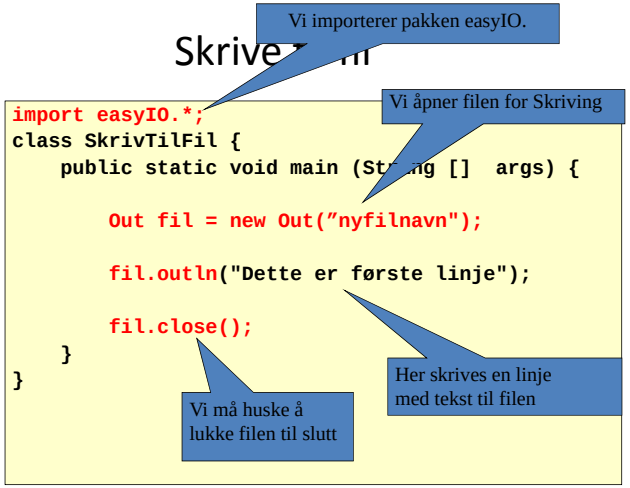
lastItem og endOfFile, lesemerket

- **endOfFile()** sjekker 'bare' om siste tegn på fila er lest
- **lastItem()** søker seg fram til første ikke-blanke tegn og returnerer **true** hvis slutten av fila nås og **false** ellers.
- Eksempel:



28

Skrive til fil



29

EasyIO: Hvilke skrivemetoder finnes?

Datatype	Eksempel	Beskrivelse
int	fil.out(x); fil.out(x, 6);	Skriv x Skriv x høyrejustert på 6 plasser
double	fil.out(x, 2); fil.out(x, 2, 6);	Skriv x med 2 desimaler Skriv x med 2 desimaler på 6 plasser
char	fil.out(c);	Skriv c
String	fil.out(s); fil.out(s, 6);	Skriv s Skriv s på 6 plasser (venstrejustert)
	fil.outln();	Skriv en linjeskift
	fil.close();	Lukk filen

Merk: dersom antall plasser spesifiseres og det ikke er plass til det som skal skrives ut, vil det som skrives ut avsluttes med tre punktumer: ...

30

Repetisjon

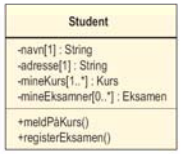
UML

UML-diagrammer

- Hvorfor tegne UML-diagrammer?
  - Oversikt
  - Samarbeid med andre programmerere/ systemutviklere
  - Arkitekter, ingeniører: tegner først, så bygger de!
  - Enklere å endre en tegning enn programme
- To typer:
  - Klassediagrammer: *domenemodell*
  - Objektdiagrammer: *sentrale datastrukturer*

32

Klassediagrammer



- Navnefeltet (alltid)
  - klassenavnet

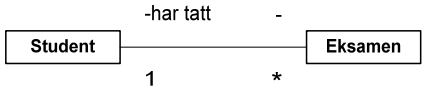
Kan utelates:

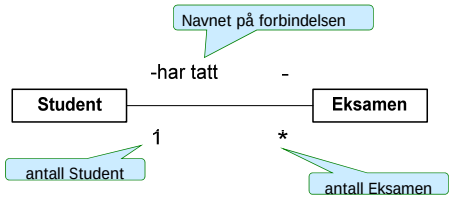
- Variabelfeltet (attributtene)
  - variabelnavn evt. med type
- Metode-feltet
  - Evt med parametere og returverdi

Symboler for synlighet
(fra resten av programmet)
+ public
- private
protected
~ package

Forhold mellom klasser

- *"En student har null eller flere eksamener…"*
- Et forhold mellom to klasser
  - Gjenspeiler en logisk kobling mellom begrepene
  - Tegnes som en strek mellom klassene
  - I koden blir dette til pekere mellom klassene som kan følges for å få tilgang til data
- Kardinalitet: vi angir hvor mange objekter det til en hver tid kan være på hver side av forholdet
- Med *eksamen* mener vi her en avlagt enkelt-eksamen, dermed vil en Eksamen bare være tilknyttet én bestemt student



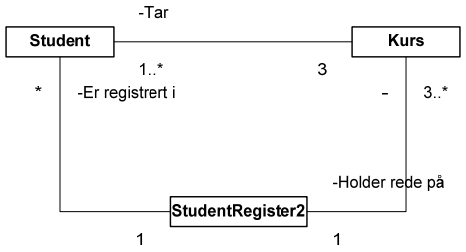


- Forbindelsen leses fra venstre:
  En student har tatt null, en eller flere Eksamener"
- Antallet objekter angis slik:

Skrivemåte	Betydning
1	en
*	null, en eller flere
1..*	minst en
3..*	minst tre
3,4,5	tre, fire eller fem

Eksempel: studentregister

- Et studentregister holder orden på studentene og kursene, og en student tar 3 kurs hvert semester



Regler for å plassere riktige antall på et forhold

1. Anta at du står i **ett** objekt av en klasse og ser over til (langs en forbindelse) til en annen klasse:
2. Hvor mange objekter ser du da maksimalt *på et gitt tidspunkt* av den andre klassen
3. Det antallet noteres (jfr. tabellen) på den andre siden
4. Du går så over forbindelsen til den andre klassen og antar at du nå står i **ett** objekt av denne klassen og gjenntar pkt. 1-3

37

Hvilke forhold skal vi ha med i klassediagrammet ?

- Slike forhold hvor ett objekt av den ene klassen:
  - inneholder
  - består av
  - eier, ...

  en eller flere objekter av den andre klassen
- I programmet vil vi følge en peker for å få tak i verdien på visse variable i den andre klassen eller kalle en metode
- Det er ikke alltid opplagt hvilke forhold vi har i et klassediagram, men det avhenger av hvilke spørsmål vi vil være interessert i å svare på

38

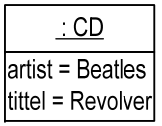
Objekt-diagrammer

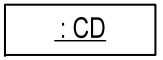
- Vi tegner en typisk situasjon av objekter i systemet vårt, når vi har fått datastrukturen på plass.
- Vi tegner og navngir bare de mest sentrale dataene som:
  - pekere
  - peker-arrayer
  - noen sentrale variable i objektene

39

Hvordan gjengis objekter?

- Ett eller to felter i en boks
- Navnfeltet
  - objektnavn:klassenavn eller bare
  - :klassenavn
- Attributt-feltet (kan være tomt)
  - Navnet på sentrale objektvariable evt. også med verdier



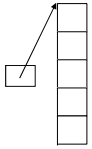


40

Andre elementer i et objektdiagram

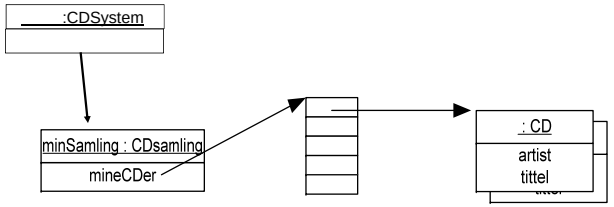
- Pekere

- Peker-arrayer



41

Eksempel: CD-system



42

Svar på noen

MOTTATTE SPØRSMÅL

Noen mottatte spørsmål

- Er det mulig å lage UML-diagrammer på forskjellige måter?
  - Ja, det er mange måter å tegne et UML-diagram på
  - Sensor er interessert i om dere har fått med de relevante klassene, og om forholdene med angivelse av antall stemmer med det oppgaven forklarer
- Hva er HashMap.values(), og fins det andre måter å lage for-løkker for HashMap?
  - HashMap.values() er en metode som returnerer en samling av verdi-objektene i en HashMap
  - Kan brukes til å løpe gjennom alle verdi-objektene med en for-løkke, se tidligere foil
  - For andre måter å lage for-løkker for HashMap, se tidligere foil
- Hva er det egentlig konstruktøren gjør, og er det viktig å ha med en konstruktør?
  - En konstruktør er en metode med *samme navn som klassen*, og *uten* returtype
  - Kan ha parametere
  - *Brukes til å initialisere objektet*
  - Hvorvidt man lager konstruktør(er) i en klasse er en skjønnsmessig vurdering
- Hva skal en array-peker brukes til? Hvordan bruker man den?
  - En array-peker er det samme som en array-variabel:

```
int[] tall;          // her opprettes array-variabelen
tall = new int[3]; // her opprettes array-objektet, og variabelen settes til å peke på det
```

  - Man bruker array-pekeren til å få tilgang til verdier i arrayen:

```
tall[0] = 1000;
System.out.println(tall[0]);
```

En større eksamensrelevant deloppgave med

HASHMAP OG METODER

Oppgave 7c – eksamen H09

- Professor Flink deklarerer klassen Restaurant som vist nedenfor. Klassen inneholder foreløpig bare deklarasjon av HashMap-variabelen kokker, så Prof. Flink ønsker din hjelp til å deklarere følgende metoder i klassen. Du kan anta at kokkenes navn skal brukes som nøkkel i HashMap'en kokker.

```
public class Restaurant {
    HashMap<String, Kokk> kokker =
        new HashMap<String, Kokk>();

    // Deklarer metodene dine her :

}
```

Metodene

- **`public boolean leggTilKokk(Kokk kokk)`**
  - Metoden tar et Kokk-objekt som parameter. Hvis det finnes et Kokk-objekt i kokker med samme navn som kokk, skal metoden returnere *false* uten å legge kokk inn i kokker. Hvis det derimot ikke finnes noe Kokk-objekt i kokker med samme navn som kokk, skal kokk legges inn i kokker og *true* returneres.
- **`public boolean fjernKokk(String navn)`**
  - Metoden skal fjerne kokken med navn **`navn`** fra kokker. Metoden skal returnere *true* dersom et Kokk-objekt ble fjernet, og *false* dersom ingen kokk med navnet navn ble funnet.
- **`public Kokk finnKokk(String navn)`**
  - Metoden skal sjekke om det finnes et Kokk-objekt med navn **`navn`** i kokker. Hvis et slikt objekt finnes, skal objektet returneres. Hvis det ikke finnes noe slikt objekt i kokker, skal *null* returneres.

Metodene (forts.)

- **`public int antallKokker()`**
  - Metoden skal returnere antall Kokk-objekter i kokker.
- **`public Kokk[] lengstAnsiennitet()`**
  - Metoden skal finne den eller de kokken(e) som har lengst ansiennitet (dvs. har vært ansatt lengst) regnet i antall år. De aktuelle Kokk-objektene skal returneres som en Kokk-array. Dersom kokker er tom, skal metoden returnere en tom Kokk-array. Dersom én kokk er alene om å ha lengst ansiennitet, skal den returnerte Kokk-arrayen ha lengde 1 og inneholde det respektive Kokk-objektet på plass 0. Hvis det derimot er n > 1 kokker som deler på å ha lengst ansiennitet, skal den returnerte Kokk-arrayen ha lengde n og inneholde alle de respektive Kokk-objektene på hver sin plass. Du kan benytte metoden ansiennitet() i klassen Kokk til å finne hvor mange år en kokk har vært ansatt.

Class Kokk

```
public class Kokk {
    private String navn ; // kokkens navn
    private int tlfnr ; // kokkens tlfnr
    private int ansettelsesAar ; // det aaret kokken ble ansatt

    public Kokk ( String navn , int tlfnr , int aar ) {
        this.navn = navn;
        this.tlfnr = tlfnr;
        this.ansettelsesAar = aar;
    }
    public String toString () {
        return " Kokk : " + navn + ", tlf " + tlfnr +
            ", ansatt " + ansettelsesAar;
    }
    public String getNavn () {
        return navn;
    }
    public int getTlfnr () {
        return tlfnr ;
    }
    public int getAnsattAar () {
        return ansettelsesAar ;
    }
    // Returnerer antall aar kokken har vaert ansatt
    public int ansiennitet () {
        Calendar cal = Calendar . getInstance ();
        return cal .get( Calendar . YEAR ) - ansettelsesAar ;
    }
}
```

leggTilKokk

```
public class Restaurant {
    HashMap<String, Kokk> kokker =
        new HashMap<String, Kokk>();

    // Deklarer metodene dine her :

    public boolean leggTilKokk(Kokk kokk) {
        Kokk funnetKokk = kokker.get(kokk.getNavn());
        if (funnetKokk != null) {
            return false;
        } else {
            kokker.put(kokk.getNavn(), kokk);
            return true;
        }
    }
}
```

fjernKokk

```
public class Restaurant {
    HashMap<String, Kokk> kokker =
        new HashMap<String, Kokk>();

    // Deklarer metodene dine her :

    public boolean fjernKokk(String navn) {
        Kokk fjernetKokk = kokker.remove(navn);
        if (fjernetKokk != null) {
            return true;
        } else {
            return false;
        }
    }
}
```

finnKokk og antallKokker

```
public class Restaurant {
    HashMap<String, Kokk> kokker =
        new HashMap<String, Kokk>();

    // Deklarer metodene dine her :

    public Kokk finnKokk(String navn) {
        return kokker.get(navn);
    }

    public int antallKokker() {
        return kokker.size();
    }

}
```

lengstAnsienitet

```
public class Restaurant {
    HashMap<String, Kokk> kokker =
        new HashMap<String, Kokk>();

    // Deklarer metodene dine her :

    public Kokk[] lengstAnsienitet() {
        HashMap<String,Kokk> lengst = new HashMap<String, Kokk>();
        int lengstAnsienitet = -1;
        for (Kokk kokk : kokker.values()) {
            if (kokk.ansienitet() > lengstAnsienitet) {
                lengst.clear();
                lengst.put(kokk.getNavn(), kokk);
                lengstAnsienitet = kokk.ansienitet();
            } else if (kokk.ansienitet() == lengstAnsienitet) {
                lengst.put(kokk.getNavn(), kokk);
            }
        }
        Kokk[] resultat = new Kokk[lengst.size()];
        int index = 0;
        for (Kokk kokk : lengst.values()) {
            resultat[index++] = kokk;
        }
        return resultat;
    }
}
```