BOKMÅL

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

KONTINUASJONSEKSAMEN I FAG
TDT4100 Objektorientert programmering / IT1104 Programmering, videregående kurs

Løsningsforslag
Merk at løsningsforslaget bare viser en måte å løse oppgavene på. For mange av oppgavene kan det være alternative løsninger.

Torsdag 16. august 2007
Kl. 09.00 – 13.00

Faglig kontakt under eksamen:
Trond Aalberg, tlf (735) 9 79 52 / 976 31 088

Tillatte hjelpemidler:

- Én og kun én trykt lærebok i Java

Sensurdato:
6. september 2007.
Resultater gjøres kjent på http://studweb.ntnu.no/ og sensurtelefon 81 54 80 14.

Prosentsatser viser hvor mye hver oppgave teller innen settet.

Merk: All programmering skal foregå i Java.

Lykke til!

Generelt for alle oppgaver

Klasser og metoder som kan være nyttige i enkelte oppgaver finner du på siste sider av eksamensoppgaven (appendiks).

Så lenge det ikke er spesifisert begrensinger for implementasjonen kan du benytte alle klasser fra Java API'en i oppgavene.

Merk at ikke alle deloppgaver krever at de foregående er løst, så ikke hopp over de resterende deloppgavene om én blir for vanskelig.

OPPGAVE 1 (15 %): Enkle metoder og klasser

a) Lag en metode `public boolean findInArray(int k, int[] a)` som returnerer true hvis verdien `k` finnes i array `a`.

```
public boolean findInArray(int k, int[] a){
      for (int i = 0; i < a.length; i ++){
            if (k == a[i]){
                  return true;
            }
      }
      return false;
}
```

b) Lag en metode `public int[] copy(int[] tab)` som returnerer en kopi av en array av int.

```
public int[] copy(int[] tab) {
      int[] copy = new int[tab.length];
      for (int i = 0; i < tab.length; i++){
            copy[i] = tab[i];
      }
      return copy;
}
```

c) Lag en klasse kalt `Rectangle` for rektangler. Et rektangel har egenskapene høyde (int), bredde (int) og farge (String). Lag en egnet konstruktør og ivareta innkapsling. Det skal være mulig å endre både høyde, bredde og farge etter instansiering. Klassen skal også ha en metode for å returnere areal `public int getArea()` (høyde x bredde).

```
public class Rectangle {

	private int height;
	private int width;
	private String color;

	public Rectangle(){
	//Lager en instans av Rectangle med default
	//høyde, bredde og farge
		this.height = 1;
		this.width = 1;
		color = "white";
	}

	public Rectangle(int height, int width, String color){
	//Lager en instans med spesifisert høyde, bredde og farge
		this.height = height;
		this.width = width;
		this.color = color;
	}

	public int getHeight() {
		return height;
	}

	public void setHeight(int height) {
		this.height = height;
	}

	public int getWidth() {
		return width;
	}

	public void setWidth(int width) {
		this.width = width;
	}

	public String getColor() {
		return color;
	}

	public void setColor(String color) {
		this.color = color;
	}

	public int getArea(){
		return width * height;
	}

}
```

d) Lag en klasse kalt `Salesman`. For en selger skal det være mulig å registrere beløp denne personen selger for ved hjelp av metoden `public void addSale(int i)`. Ved hjelp av metoden `public int[] getSales()` skal det returneres et array med de beløpene denne personen har solgt for, og metoden `public int getTotal()` skal returnere det totale beløp denne personen har solgt for.

```
import java.util.ArrayList;
import java.util.List;

public class Salesman {

	//List av Integer er enklere å bruke enn tabell
	private List<Integer> sales;

	public Salesman(){
		sales = new ArrayList<Integer>();
	}

	public void addSale(int i){
		sales.add(i);
	}

	public int[] getSales(){
	//Kopier til tabell
	//Ikke bruk List.toArray() siden denne vil returnere
	//en tabell av Integer og ikke int
		int[] salestab = new int[sales.size()];
		for (int i = 0; i < sales.size(); i ++){
			salestab[i] = sales.get(i);
		}
		return salestab;
	}

	public int getTotal(){
		int total = 0;
		for (Integer i: sales){
			total += i;
		}
		return total;
	}

}
```

e) Finn og rett feil i denne koden:

```
public class Person {

	//1: fjern static modifier for name-felt
	//navn er et typisk instans-felt og ikke klasse-felt

	private String name;

	//2: fjern void fra konstruktør
	//Konstruktører deklareres uten retur-verdi
	public Person(String name){

		//3: bytt om broken av this;
		//this statment sets the changes the parameter value
		this.name = name;
	}

}
```

OPPGAVE 2 (35%): Klasse for lottokupong

I denne oppgaven skal du lage en klasse for lottokuponger. En lottokupong inneholder et ukenummer som identifiserer trekningen kupongen gjelder for (heltall mellom 1 og 52) og lottokupongen består ellers av opp til 10 rader med tall. Hver rad skal inneholde 7 forskjellige heltall mellom 1 og 34. Alle tall i en rad er unike. Kall klassen din for `LotteryTicket.` I denne oppgaven skal du definere klassens felt(er), lage konstruktører og andre metoder.

`NB!` Alle svar skal være i form av kode og evt. forklarende tekst eller begrunnelser. Oppgaven kan løses steg for steg i henhold til oppgaveteksten, men du kan også lage en samlet implementasjon av hele klassen. Hvis du mener det forenkler koden din kan du godt lage egne hjelpemetoder eller andre klasser.

a) Definer klassen med de nødvendige feltene (du trenger ikke å lage metoder ennå). Forklar hvordan vil du implementere felt(et) for radene med tall? Før du velger type felt er det lurt å se over de øvrige av oppgavens deloppgaver slik at du velger en egnet type. Tips: du bør bruke klasser fra Java sitt Collection rammeverk; for eksempel ArrayList, LinkedList, TreeSet (sett med ordnet rekkefølge), HashSet (sett som ikke er ordnet).

```
public class LotteryTicket {

      //Oppgaven er enkel å løse hvis du bruker en liste av set
      //eller liste av liste. Fordelen med å bruke List-type i stedet
      //for vanlig tabell er at du slipper å endre størrelse
      //Fordelen med å bruke liste av set er at det forenkler generering
      //av et sett med unike nummer i en seinere oppgave

      List<Set <Integer>> rows;
      int week;
      Random rand = new Random();
}
```

b) Lag en konstruktør som tar ukenummer (int) som parameter. Når denne kalles skal det opprettes en kupong hvor ukenummer er satt og som er klar for innelegging av rader av tall.

```
      //Det viktigste her er å huske å bruke this hvis felt og
      //parameter har samme navn, samt å initialisere evt. liste
      public LotteryTicket(int week){
            this.week = week;
            rows = new ArrayList<Set<Integer >>();
      }
```

c) Lag en metode for å legge til rader av tall: `public void addRow(int[] row)`. Det skal ikke være mulig å erstatte rader som allerede er lagt til og det skal ikke være mulig å legge til flere enn 10 rader. I denne deloppgaven kan du anta at `row` er et gyldig sett av tall (dvs. 7 unike tall mellom 1 og 34), men du kan ikke anta at tallene er i stigende rekkefølge (sortert). Det skal være mulig å legge til to rader som har de samme tallene.

```
	//Her brukes en hjelpemetode makeRow(int[] row) for å lage et sett
	public void addRow(int[] row){
		if (rows.size() < 10){
			rows.add(makeRow(row));
		}
	}

	private Set<Integer> makeRow(int[] row){
		Set<Integer> rowset = new TreeSet<Integer>();
		for (int i = 0; i < row.length; i ++){
			rowset.add(row[i]);
		}
		return rowset;
	}
```

d) Lag en konstruktør som tar et ukenummer og en boolean som parameter. Hvis boolean parameter er false skal det opprettes en tom kupong som i deloppgave b), hvis parameter er true skal det opprettes en utfylt kupong hvor alle 10 radene er fylt ut med tilfeldig valgte tall. Du kan bruke klassen java.util.Random for å velge tall tilfeldig (se vedlegg).

```
	//Her bruker vi hjelpemetoden makeRow() for å lage
	//en tilfeldig valgt rekke ved hjelp av Random-klassen
	public LotteryTicket(int week, boolean filledIn){
		//Husk å kale annen konstruktør i første linje
		this(week);
		if(filledIn){
			for (int i = 0; i < 10; i ++){
				rows.add(makeRow());
			}
		}

	}

	public Set<Integer> makeRow(){
		Set<Integer> rowset = new TreeSet<Integer>();
		while (rowset.size() <= 7){
			rowset.add(rand.nextInt(34) + 1);
		}
		return rowset;
	}
```

e) Lag en metode `public int getResult(int rownum, int[] draw)` som du kan bruke for å sjekke om du har vunnet. Parameteren `rownum` er raden i kupongen det skal sjekkes mot og parameteren `draw` er et array av vinnertallene i en trekning. Det er premie for 5, 6 og 7 rette tall. Metoden skal returnere tallet 1 hvis det er 7 rette tall, 2 hvis det er 6 rette, 3 hvis det er 5 rette og 0 hvis du har mindre enn 5 rette tall. I denne deloppgaven kan du anta at `draw` er et gyldig sett av tall (dvs. 7 unike tall mellom 1 og 34), men du kan ikke anta at tallene er i stigende rekkefølge (sortert).

```
public int getResult(int rownum, int[] draw){
	int result = 0;
	for (int i = 0; i < draw.length; i ++){
	//bruk contains() eller iterer for å sjekke om verdi finnes
		if (rows.get(rownum).contains(draw[i])){
			result ++;
		}
	}
	if (result == 7){
		return 1;
	}else if (result == 6){
		return 2;
	}else if (result == 5){
		return 3;
	}else{
		return 0;
	}
}
```

f) Skriv kode for en metode `public String toString()` som returnerer en String med informasjonen fra en lottokupong. Trekningsuke skal være på første linje, deretter skal alle radene med tall være på en linje hver på formen ”1, 2, 5, 8, 23, 24”. Merk at det ikke er komma etter siste tall.

```
//Enklest å lose hvis du bruker iterator
//Da har du muligheten til å sjekke om det finnes flere elementer
//og dermed avgjøre om det skal være komma eller ikke mellom
//tallene i utlistingen
public String toString(){
	String result = "";
	for (Set<Integer> s: rows){
		Iterator<Integer> it = s.iterator();
		while (it.hasNext()){
			result += it.next();
			if (it.hasNext()){
				result += ", ";
			}
		}
		result += "\n";
	}
	return result;
}
//Hvis du har brukt List for tallrekkene kan du sortere med
//Collections.sort()for å få en sortert liste
//Hvis du har brukt array kan du benytte Arrays.sort()
```

g) I oppgave b) er det beskrevet en addRow-metode hvor vi antar at raden med tall som oversendes i parameteren er gyldig (dvs. 7 unike tall mellom 1 og 34). Lag en unntaksklasse kalt InvalidRowException og lag en ny versjon av addRow som kaster unntak av denne typen hvis man prøver å legge til ugyldige rader av tall; for eksempel som inneholder for få tall eller at samme tall er repetert. Du kan anta at det allerede finnes en metode for å sjekke om en rad er gyldig eller ikke: `public boolean isValidRow(int[] row)`.

```
public class InvalidRowException extends Exception {

	public InvalidRowException(){
		super();
	}

	public InvalidRowException(String msg){
		super(msg);

	}

}

//I LotteryTicket:



//Denne metoden er kun tatt med for å vise hvordan den kunne vært
//Studentene skal ikke implementere denne, bare bruke den
public boolean isValidRow(int[] row){
	//Using makeRow(row) to make an ordered set
	Set<Integer> temprow = makeRow(row);
	//testing for valid number of numbers
	if (temprow.size() != 7){
		return false;
	}
	//tesing for valid numbers and sequence
	int i = 0;
	for (Integer j: temprow){
		if (j != row[i++]){
			return false;
		}
	}
	return true;
}


public void addRow (int[] row) throws InvalidRowException{
	if (isValidRow(row)){
		if (rows.size() < 10){
			rows.add(makeRow(row));
		}
	}else{
		throw new InvalidRowException("Invalid row");
	}
}
```

h) Du vil at alle lottokuponger skal få et unikt nummer ved instanisering (int id). Vis ved hjelp av kode og forklaring hvordan du kan implementere dette i `LottoTicket` klassen.

```
//felter:
private static int counter = 0;
private int number;

//I konstruktør:
this.number = ++counter;

//metode for å lese nummer på lottokupong
public int getNumber(){
      return number;
}
```

OPPGAVE 3 (20 %): Iteratorer

a) Hva er en iterator og hvilke 2 metoder er det som er karakteristiske for iteratorer?

```
//En iterator er et objekt som brukes til å iterere over elementene
//i en samling, eller til å returnere sekvenser av objekter

public boolean hasNext()
public Object next()
```

b) Ta utgangspunkt i iterator-variabelen `Iterator<Person> it` og lag en while-løkke som skriver ut navnene på alle personer vha. `Person`-klassens `getName`-metode.

```
Iterator<Person> it = c.iterator();
while (it.hasNext()){
      System.out.println(it.next().toString());
}
```

c) Lag en Iterator-klasse kalt `MergeIt` som implementerer Java-grensesnittet `Iterator` og som tar to objekter av typen List som parameter i konstruktøren `MergeIt(List list1, List list2)`. `MergeIt` skal returnere en flettet sekvens av objekter fra de to listene: første gang returneres objekt 1 fra list1, andre gang returneres objekt 1 fra list2, tredje gang objekt 2 fra list1, fjerde gang objekt 2 fra list2 etc. Hvis en liste er lengre enn den andre så avsluttes sekvensen med de siste elementene fra den lengste lista slik iteratoren returnerer alle objekter fra begge lister. NB! Du trenger kun å implementere de to metodene som er karakteristiske for iteratorer.

```
// I denne oppgaven skal dere vise at dere kan å implementere next- og
hasNext-metodene
//Det beste alternativet er å bruke iteratorene fra de to listene og
bytte om hvilken det hentes fra for hvert kall (for eksempel med testing
på om det er flere igjen i den andre)
//En alternativ løsning (som ikke er fullt så god) er selvsagt å lage seg
en ny flettet liste med innholdet fra de to list1 og list2

public class MergeIt implements Iterator{

	Iterator current;
	Iterator other;

	public MergeIt(List list1, List list2){
		current = list1.iterator();
		other = list2.iterator();

	}

	public boolean hasNext() {
		return current.hasNext() || other.hasNext();
	}

	public Object next() {
		Object tempObject = null;;
		if (current.hasNext()){
			tempObject = current.next();
		}
		if (other.hasNext()){
		//switching current with other if other has more objects
			Iterator tempIt = current;
			current = other;
			other = tempIt;
		}
		return tempObject;
	}

}
```

OPPGAVE 4 (30%): Klasser, arv og grensesnitt

I denne oppgaven skal du implementere en avtalebok. En avtalebok kan inneholde mange forskjellige typer hendelser og i denne oppgave skal du lage klasser for hendelser som er forskjellig med hensyn til antallet dager de gjelder for.

a) Implementer følgende typer: `Event`, `SingleDayEvent`, `ManyDaysEvent`, `ReoccuringEvent`.

PS! Alle datoer i denne oppgaven kan implementeres som `String` og du kan anta at alle datoer er på samme format. Husk å implementere felter og metoder som er nødvendig for å sette og hente dato(er).

`Event` er den generelle typen for alle hendelser. Du må bestemme selv om du skal bruke en klasse, abstrakt klasse eller grensenitt for denne. Alle hendelser har det til felles at de har et innkapslet `Subject`-felt. Alle hendelser skal også ha metoden `public boolean onDate(String date)` som returner true hvis denne hendelsen finner sted på datoen i parameteren.

```
public abstract class Event{

	//Bruk abstract klass med onDate som abstract metode

	private String subject;

	public Event(){
		this.subject = "no subject";
	}

	public Event(String subject){
		this.subject = subject;
	}

	public String getSubject() {
		return subject;
	}

	public void setSubject(String subject) {
		this.subject = subject;
	}

	public abstract boolean onDate(String date);

}
```

En instans av `SingleDayEvent` har en enkelt dato og gjelder kun for denne datoen. For objekter av denne typen skal onDate-metoden returnere true hvis datoen er den samme som date-parameteren.

```
public class SingleDayEvent extends Event {

	String date;

	public SingleDayEvent(String subject, String date){
		super(subject);
		this.date = date;

	}

	public String getDate() {
		return date;
	}


	public void setDate(String date) {
		this.date = date;
	}


	public boolean onDate(String date) {
		return this.date.equals(date);
	}

}
```

En instans av `ManyDaysEvent` har en startdato og en sluttdato. For objekter av denne typen skal onDate-metoden returnere true hvis dato er større enn eller lik fradato og mindre enn eller lik sluttdato. Tips: du kan bruke compareTo-metoden fra Comparable-grensesittet som String-klassen implementer for å sjekke om en dato er større, mindre eller lik.

```
public class ManyDaysEvent extends Event {


	private String fromDate;
	private String toDate;

	public ManyDaysEvent(String subject, String fromDate, String toDate){
		super(subject);
		this.fromDate = fromDate;
		this.toDate = toDate;
	}

	public String getFromDate() {
		return fromDate;
	}

	public void setFromDate(String fromDate) {
		this.fromDate = fromDate;
	}

	public String getToDate() {
		return toDate;
	}

	public void setToDate(String toDate) {
		this.toDate = toDate;
	}

	public boolean onDate(String date) {
		return ((this.fromDate.compareTo(date) <= 0) &&
		(this.toDate.compareTo(date) >= 0));
	}
}
```

En instans av `ReoccuringEvent` har flere forskjellige datoer og gjelder kun for enkeltdagene som er lagt inn for denne. For objekter av denne typen skal onDate-metoden returnere true hvis dato finnes blant datoene som lagret for denne hendelsen.

```
public class ReoccuringEvent extends Event {

	private List<String> dates;

	public ReoccuringEvent(String subject, List<String> dates){
		super(subject);
		this.dates = new ArrayList<String>();
		//making a copy of values
		for (int i = 0; i < dates.size(); i ++){
			this.dates.add(dates.get(i));
		}
	}

	public int getSize(){
		return dates.size();
	}

	public String getDate(int i){
		return dates.get(i);
	}

	public void addDate(String date){
		dates.add(date);
	}

	public boolean onDate(String date) {
		for(String s: dates){
			if (s.equals(date)){
				return true;
			}
		}
		return false;
	}

}
```

b) Implementer klassen `Calender`. Denne klassen skal kunne lagre forskjellige typer hendelser og ha en metoden `public void listEvents(String date)` som finner og skriver ut subject for alle hendelser som gjelder for denne dagen.

```
public class Calender {

	private ArrayList<Event> events = new ArrayList<Event>();

	public void addEvent(Event ev){
		events.add(ev);
	}

	public void listEvents(String date){
		for (Event e: events){
			if (e.onDate(date)){
				System.out.println(e.getSubject());
			}
		}
	}

}
```

c) Vis hvilke endringer du må gjøre i klassene dine og lag en metode `public List<Event> findEvents(String date)` som returnerer en liste over hendelser for angitte dato. Listen som returneres skal være sortert i stigende rekkefølge på subject. *Du kan anta at det finnes en implementasjon av List-grensesnittet kalt SortedList som støtter sortert liste gitt objekter gitt de samme vilkårene som gjelder for eksempel for sorterte Set- eller Map-implementasjoner.*

```
//Endringer i Event: implementer Comparable grensesnittet
//og metoden compareTo

public abstract class Event implements Comparable<Event>{

	public int compareTo(Event event) {
		return this.subject.compareTo(event.getSubject());
	}

}

//I Calender-metoden findEvents:
//Bruk java.util.Collections.sort(List l)
//Sortering eller ordning gjører i Collection-rammeverket
//vha. av Comparable-grensesnittet; dvs. i praksis at objekter som
//implementerer dette kan sorteres eller sammenlignes

public List<Event> findEvents(String date){
	List<Event> foundevents = new ArrayList<Event>();
	for (Event e: events){
		if (e.onDate(date)){
			foundevents.add(e);
		}
	}
	java.util.Collections.sort(foundevents);
	return foundevents;
}

//PS! For at dere skulle slippe å tenke sorteringsalgoritme ga
//oppgaven mulighet for å bruke den fiktive klassen SortedList
//Da blir løsningen som følger:

public List<Event> findEvents(String date){
	List<Event> foundevents = new SortedList<Event>();
	for (Event e: events){
		if (e.onDate(date)){
			foundevents.add(e);
		}
	}
	return foundevents;
}
// Oppegående studenter la kanskje lagt merke til at dette vil
//være litt ulogisk å ha en slik subtype av Liste siden det er
//meningsløst å bruke posisjonsbasert innsetting og uthenting på
//når plassering er basert på en sortert rekkefølge.
```

d) Hva betyr det hvis du deklarerer en klasse til å være final?
Hva betyr det hvis du deklarerer en metode til å være final?

```
//Klasser som er final kan ikke subklasses, dvs det er ikke lov
//å lage klasser som arver disse

//Metoder som er final kan ikke redefineres i klasser som arver
```

Appendiks

Klasser og metoder som kan være nyttige:

ArrayList-klassen:

E	**get**(int index) Returnerer elementet på en gitt plass i lista
boolean	**add(E e)** **Legger til element på slutten av lista Appends the specified element to the end of this list.**
boolean	**contains(Object o)** **Returnerer true hvis lista inneholder det spesifiserte elementet.**
E	**get(int index)** **Returnerer elementet på den spesifiserte poisjonen i lista.**
Iterator<E>	**iterator()** **Retunerer en iterator over elementene i lista.**

TreeSet-klassen:

boolean	**add(E e)** Legger elementet til dette settet hvis det ikke fines fra før.
Iterator<E>	**iterator()** Returnerer en iterator som gir deg elementene i sortert rekkefølge
boolean	***contains(Object o)*** *Returnerer true hvis lista inneholder det spesifiserte elementet.*

Metoder i Random-klassen

int	**nextInt**() Returnerer en tilfeldig valgt int verdi
int	**nextInt**(int n) Gir deg en tilfeldig valgt int verdi som er >= 0 og < n.

Metoder i Comparable-grensesnittet:

int	**compareTo**(T o) Sammenlinger dette objektet med det spesifiserte objektet. Returnerer et negivt integer, null eller eller positivt integer hvis dette objektet er mindre enn, lik eller større enndet spesifiserte objektet.