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

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`.

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

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).

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.

e) Finn og rett feil i denne koden:

```
public class Person {

      private String name;

      public Person(String name){
            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).

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.

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.

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).

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).

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.

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)`.

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.

OPPGAVE 3 (20 %): Iteratorer

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

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.

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.

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.

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.

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.

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.

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.

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.*

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?

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.