NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

KONTINUASJONSEKSAMEN I FAG
TDT4100 Objektorientert programmering

BOKMÅL

Onsdag 16. august 2006
Kl. 09.00 – 13.00

Faglig kontakt under eksamen:
Trond Aalberg, tlf (735)97952 / 976 31088

Tillatte hjelpemidler:

- Én og kun én trykt bok, f.eks. Lewis & Loftus: Java Software Solutions

Sensurdato:
6. september 2006.
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!

OPPGAVE 1 (20%): Metoder og kontrollstrukturer

Klasser og metoder som kan være nyttige for denne oppgaven finner du på siste side av eksamensoppgaven.

a) Skriv kode for en metode `String stringMulti(String str, int x)` som returnerer en ny streng bestående av strengen `str` gjentatt `x` ganger. Eksempelvis skal metodekallet `stringMulti("ah", 4)` returnere strengen *"ahahahah"*.

b) Skriv kode for en metode `String swapChars(String str, int p1, int p2)` som returnerer en ny `String` hvor tegnene i indeks p1 og p2 er byttet om. Eksempelvis skal `swapChars("abcde", 2, 4)` returnere strengen "*abedc*".

c) Et palindrom er et ord eller en setning som har samme mening enten en begynner å lese forfra eller bakfra. Eksempler på palindromer er "*agnes i senga*", "*regninger*" og "*grav ned den varg*". Et palindrom kan ha et antall bokstaver som enten er partall eller oddetall og det er uvesentlig om det er brukt store bokstaver eller små bokstaver. Du skal med andre ord i løsningen ta høyde for at både "*abcba*" og "*Abccba*" er palindromer. For palindromer er også mellomrom en del av strengen.

   Skriv kode for en metode `boolean isPalindrom(String str)` som returnerer true hvis strengen er et palindrom og false hvis den ikke er et palindrom.

d) Et ord/uttrykk som er et anagram for et annet ord/uttrykk består av de samme bokstavene men rekkefølgen er byttet om og meningen er forskjellig. Eksempelvis er "*amor*" et anagram for "*Roma*", "*Tom Marvolo Riddle*" er et anagram for "*I am Lord Voldemort*", og "*Presbyterians*" er et anagram for " *Britney Spears*". Legg merke til at anagrammer kan være forskjellige fra det opprinnelige ordet med hensyn til blanke tegn og store/små bokstaver, men det er det samme sett av bokstaver som er brukt i begge.

   Skriv metoden `boolean areAnagrams(String str1, String str2)` som returnerer true hvis strengene er anagrammer og false hvis de ikke er anagrammer.

OPPGAVE 2 (40%): Klasser

I denne oppgaven skal du lage klasser for å beskrive et hus med etasjer og rom, og du skal løse forskjellige deloppgaver i tilknytning til denne enkle objekt-modellen.
Merk at ikke alle deloppgaver krever at de foregående er løst, så ikke hopp over de resterende deloppgavene om én blir for vanskelig. Bruk gjerne grensesnitt og klasser fra Java sitt API/klassebibliotek, f.eks. Collection-rammeverket.

a) Definer klassene Hus, Etasje og Rom. Du kan utelate konstruktører og metoder i denne delen av oppgaven siden du skal gjøre dette i de neste deloppgavene.
   - Et rom har et areal (heltall) og har en betegnelse (f. eks. ”bad”, ”stue”, ”kjøkken”).
   - Etasjer har et navn (”kjeller”, ”første”, ”andre”, ”loft”).
   - Et hus har en adresse og en eier.
   - Det kreves at klassene dine er i stand til å holde rede på hvilke rom som finnes i hvilke etasjer og hvilke etasjer et hus inneholder.

b) For klassen Hus skal det være mulig å kalle metoden `public int getBoligAreal()` for å få returnert boligarealet i hele huset. Lag denne metoden og evt. andre metoder du mener er nødvendige for å støtte denne funksjonaliteten. NB! Bruk metoder og tenk objekt-orientert.

c) Du ønsker å lage kode som gjør at det *ikke* skal være mulig å instansiere rom uten at disse knyttes til en etasje og at det *ikke* skal være mulig å instansiere etasjer uten at disse knyttes til et hus. Forklar og vis med kode hvordan du kan sikre at bare objekter med gyldig tilstand skal kunne instansieres. Hint: vi er interessert i hvordan du vil implementere klassenes konstruktør(er) for å oppnå dette.

d) Forklar og vis med kode hvordan du kan sikre at et rom kun er assosiert til *en* etasje og hvordan du kan sikre at et et hus ikke inneholder flere etasjer med samme navn.
Hint: i denne oppgaven kan du bygge videre på deloppgave c).

e) Du ønsker å kunne støtte automatisk nummerering av rom-objekter etter hvert som de instansieres. Det første rom-objektet som instansieres skal få nummer 1, det andre objektet 2 etc. Forklar og vis med kode hvordan du kan implementere dette i Rom-klassen (og kun i denne klassen).

f) Hva er unntakshåndtering (exceptions)? Forklar kort når du bør bruke unntakshåndtering (her er vi ute etter generelle regler, men inkluder gjerne eksempler).

g) Hvis et program prøver å legge samme rom-objekt til to etasjer skal det kastes et unntak. Vis ved hjelp av kode hvordan du vil deklarere en unntaksklasse for denne situasjonen og forklar med tekst eller kode hvordan du ville benyttet denne i koden fra oppgave d).

h) I tillegg til hus ønsker du at objektmodellen din skal støtte objekter av typen leilighet. En leilighet kan på samme måte som hus inneholde flere etasjer, men en leilighet vil bestandig være del av et hus. Lag en superklasse Bolig som har Hus og Leilighet som subklasser og forklar hvilke metoder/felter du vil ha i superklassen og hvilke du vil ha i subklassen.

OPPGAVE 3 (30%): Grensesnitt og abstrakte klasser

I denne oppgaven skal du implementere forskjellige figurklasser og vise og forklare hvordan grensesnitt og abstrakte klasser kan benyttes.

a) Hva er et grensesnitt (interface) og hva bruker vi grensesnitt til i programmering?
Lag et Java interface (`interface`) kalt `FigurGrensesnitt`. Objekter av denne typen skal tilby metoder som returnerer fargen, arealet og typen til en figur ved hjelp av metodene `public String getFarge()`, `public double getAreal()` og `public String getType()`. Med type mener vi her en tekststreng som beskriver hva slags figur det er (*"firkant"*, *"rektangel"* eller *"sirkel"*).

b) Lag klassene Rektangel og Sirkel som begge implementerer grensesnittet `FigurGrensesnitt`. Følgene metoder skal også implementeres i klassene:

Rektangel: `public int getHøyde()` og `public int getBredde()`
Sirkel: `public int getRadius()` og `public double getOmkrets()`

Alle felter skal være innkapslet og det skal *kun* være mulig å sette objektenes verdier ved instansiering.

PS! For den som ikke husker formlene for en sirkels omkrets og areal så er omkrets = $2*\pi*r$ og arealet = $\pi*r^2$. Tips: Du kan benytte `Math.PI` i Java.

c) Lag en abstrakt klasse AbstraktFigurImpl som implementerer FigurGrensesnitt og som Rektangel og Sirkel er subklasser av. Forklar hvordan AbstraktFigurImpl, Rektangel og Sirkel bør være mht. metodene og feltene som er beskrevet i deloppgavene over. Også i denne deloppgaven skal alle felter være innkapslet og det skal kun være mulig å sette verdier ved instansiering.

d) I denne oppgaven skal du lage `toString`-metoder som kan brukes for å få skrevet ut informasjon om figur-objekter. For alle objekter skal farge, type og areal skrives ut. I tillegg skal høyde og bredde skrives ut for rektanglar mens det for sirklar skal skrives ut radius og omkrets. Utskriften for hhv. sirkler og rektangler skal se slik ut:

"rød sirkel, areal = 78.53, radius = 5, omkrets =31.41"
"grønn rektangel, areal = 12, høyde = 3, bredde = 4"

Forklar og vis med kode hvordan du implementerer `toString`-metoder i den abstrakt klassen og i de konkrete figurklassene (Rektangel og Sirkel) slik at den abstrakte klassen har ansvar for å lage første del av teksten (eks. "*rød sirkel, areal 78.53*") mens de konkrete klassene har ansvaret for å legge til infomasjonen som er spesifikk for hver klasse (for eksempel radius og omkrets for sirkler).

Vis bruk av `toString`-metoden ved at du lager en enkel main-metode hvor du oppretter en tabell (array) av figurer og bruker ei løkke for å skrive ut informasjon om figurene.

OPPGAVE 4 (10%): Regler for oppførsel og testing

a) Definer presise og etterprøvbare regler for oppførselen til metoden som er beskrevet i oppgave 1 d) : `boolean areAnagrams(String str1, String str2).`
I tillegg til den beskrivelsen som finnes i oppgave 1 d) skal du også definere andre regler du mener er aktuelle for en sikker og robust metode (her menes regler som kan utledes fra det som er skrevet om metoden, eller regler som du selv mener er relevant).

b) Skriv en eller flere testmetoder i en tenkt TestCase-subklasse med JUnit-rammeverket, som til sammen tester `areAnagrams`-metoden. Relevante metoder fra TestCase-klassen er: `assertEquals(Object, Object)`, `assertTrue(boolean)`, `assertFalse(boolean)` og `assertNull(Object)`.

Appendiks

Klasser og metoder som kan være nyttige i oppgave 1:

Metoder i `String`-klassen:

```
String(char[] value)
// Allocates a new String so that it represents the sequence of
// characters currently contained in the character array argument.

char charAt(int index)
// Returns the char value at the specified index.

int length()
//Returns the length of this string.

replaceAll(String regex, String replacement)
// Replaces each substring of this string that matches the given regular
// expression with the given replacement.
// Denne metoden fungerer i prakis som en søk og erstatt metode og kan
// benyttes til å erstatte en substreng (regex) med en annen streng
// (replacement) f.eks. vil replaceAll("og", "eller") erstatte alle
// forekomster av "og" med "eller". Husk at en streng også kan være ett
// enkelt tegn eller en tom streng.

char[] toCharArray()
// Converts this string to a new character array.

String toUpperCase()
// Converts all of the characters in this String to upper case.

String toLowerCase()
// Converts all of the characters in this String to lower case.

String concat(String str)
// Concatenates the specified string to the end of this string
// (på norsk: metoden legger til den spesifiserte strengen på slutten av
// strengen).
```

Metoder i Arrays-klassen

```
static boolean equals(char[] a, char[] a2)
// Returns true if the two specified arrays of chars are equal to one
another.

static void sort(char[] a)
//Sorts the specified array of chars into ascending numerical order.
```