NTNU
Institutt for datateknikk
og informasjonsvitenskap

Eksamensoppgave i

TDT4100 – Objektorientert programmering

Lørdag 22. mai 2010, kl. 09:00 - 13:00

Oppgaven er utarbeidet av faglærer Hallvard Trætteberg og kvalitetssikrer Trond Aalberg.
Kontaktperson under eksamen Hallvard Trætteberg (mobil 918 97263)

Språkform: Bokmål
Tillatte hjelpemidler: C
Én valgfri lærebok i Java er tillatt.
Bestemt, enkel kalkulator tillatt.

Sensurfrist: Tirsdag 15. juni.

Les oppgaveteksten nøye. Finn ut hva det spørres etter i hver oppgave.

Dersom du mener at opplysninger mangler i en oppgaveformulering gjør kort rede for de antagelser og forutsetninger som du finner det nødvendig å gjøre.

Del 1 – Innkapsling (20%)

a) Hva er hensikten med innkapsling?

b) Lag en **`Date`**-klasse med metoder for å lese og endre informasjon om *dag*, *måned* og *år*, slik at denne informasjonen blir ordentlig *innkapslet* og *validert*. Velg selv konstruktør(er) og metoder som inngår i innkapslingen, egnet intern representasjon og evt. interne hjelpemetoder. Du kan anta det finnes en hjelpemetode som sier om et år er et skuddår. Gjør kort rede for dine valg og husk at det finnes flere fullgode løsninger.

c) Implementer metoder for å endre datoen til neste og forrige dag. Merk at disse metodene skal endre **`Date`**-objektet selv, ikke lage nye **`Date`**-objekter.

Del 2 – Typer (10%)

Anta følgende klasser, grensesnitt og metoder:
- Klasse **`A`** deklarerer metoden **`A methodA()`**.
- Grensesnittet **`G`** deklarerer metoden **`G methodG(A)`** .
- Klasse **`B`** arver fra **`A`**, implementerer **`G`** og deklarerer metoden **`B methodB(A)`**.
- Klasse **`C`** implementerer **`G`** og deklarerer metoden C **`methodC(G)`**.

a) Hvilke av følgende deklarasjoner/initialiseringer vil gi feil i editoren/ved kompilering:

```
A a = new B();
B b = new A();
G g1 = new A(), g2 = new B(), g3 = new C();
```

b) Anta i det følgende at variablene **`a`**, **`b`** og **`c`** er deklarert til å være av klassene **`A`**, **`B`** og **`C`**. Hvilke av følgende uttrykk vil gi feil i editoren/ved kompilering og hvorfor:

```
a.methodA() == c.methodC(a)
b.methodA().toString()
b.methodG(new B())
((G) a).methodG(a)
((B) a).methodG(b)
((C) a).methodG(a)
```

Del 3 – Klasser (35%)

For å kunne stave ord over en dårlig kommunikasjonslinje er det vanlig å bruke et ord-basert alfabet, hvor første bokstav i ordet tilsvarer bokstaven som sies. Nato har standardisert dette i sitt fonetiske alfabet, som begynner med ”alfa”, ”bravo”, ”charlie”, ”delta”, ”echo”, ”foxtrot”. Dersom ordet ”java” staves med dette alfabetet blir det ”juliet alfa viktor alfa”.

Du skal implementere en klasse **`RadioAlphabet`** for å konvertere bokstaver og ord vha. denne teknikken. Du velger selv hvordan alfabetet representeres, f.eks. kan en **`String`**, **`String`**-tabell, **`List`** eller **`Map`** brukes for å representere alfabetet. Merk at det kan være lurt å lese gjennom hele oppgaven og ta en titt på oppgave 5, før du velger mellom en av disse og evt. andre alternativer.

a) Først skal du implementere én konstruktør og tre metoder, som følger:

- Konstruktøren skal ta inn ordene i alfabetet som en **`String`** og en **`String`** som inneholder skilletegnet. Merk at ordene trenger ikke være i alfabetisk rekkefølge, fordi første bokstav i ordet avgjør hvilken bokstav det tilsvarer. F.eks. skal **`new RadioAlphabet(”alfa-bravo-delta-charlie”, ”-”)`** gi et objekt som kan oversette bokstavene ’a’-’d’ (og bare disse) med Nato-alfabetet. Alfabetet skal kun kunne inneholde ord for bokstaver (altså ikke tall og skilletegn).

- Metoden **`boolean converts(char)`** skal returnere om tegnet er med i alfabetet, dvs. har en oversettelse.

- Metoden **`String convert(char)`** skal returnere ordet som tilsvarer tegnet som gis som parameter, eller **`null`** dersom tegnet ikke kan oversettes. F.eks. skal **`convert('a')`** returnere **`”alfa”`** og **`convert('!')`** skal returnere **`null`**.

- Metoden **`String convert(String)`** skal returnere sekvensen av ord i en **`String`** som tilsvarer ordet som gis som parameter. Tegn som ikke kan oversettes skal ignoreres. Det skal være ett mellomrom mellom ordene. F.eks. skal **`convert(”*ab*ba*”)`** returnere **`”alfa bravo bravo alfa”`**.

b) Hvorfor kan ikke disse metodene være deklarert med **`static`**-modifikatoren?

c) Hva kalles det når metoder i samme klasse har samme navn (som **`convert`**-metodene)? Hva er regelen for å skille dem fra hverandre/velge mellom dem, når de brukes/kalles?

d) Utvid klassen med metodene **`setWord(String)`** som endrer ordet som brukes for en bestemt bokstav (første bokstav i ordet), og **`removeWord(char)`** som fjerner ordet for en bestemt bokstav fra alfabetet. Merk at **`setWord`** skal kun kunne endre ordet for en bokstav som allerede finnes i alfabetet.

Del 4 – Testing (10%)

a) Skriv en JUnit-testklasse som tester begge **`convert`**-metodene i **`RadioAlphabet`**-klassen. Du kan anta at konstruktøren virker. Vi er ikke så nøye på detaljene, bare den generelle testeteknikken er riktig.

b) Forklar hvordan du vil teste **`setWord`**- og **`removeWord`**-metodene i **`RadioAlphabet`**-klassen vha. JUnit-teknikken. Hva er en vesentlig (og kompliserende) forskjell mellom å teste disse metodene og å teste **`convert`**-metodene?

Del 5 – Arv og observerbarhet (25%)

a) Morse-alfabetet er et alfabet hvor bokstaver representeres med sekvenser av prikker og streker. F.eks. skrives det internasjonale nødsignalet "SOS" som "... --- ..." fordi 's' skrives som "..." og 'o' som "---". Du skal lage en subklasse av **`RadioAlphabet`**-klassen kalt **`MorseAlphabet`**, som støtter dette alfabetet. F.eks. skal **`new MorseAlphabet()`** gi et ferdig-initialisert objekt som kan konvertere **`"sos"`** til **`"... --- ..."`** og **`"test"`** til **`"- . ... -"`**. **`convert`**-metodene og **`removeWord`** skal virke som spesifisert i del 3. **`setWord`**-metoden må nødvendigvis endres slik at den tar inn tegnet som første argument og ordet som andre, f.eks. **`setWord('s', "...")`**.

Det er et mål å utnytte arv slik at det blir mest mulig gjenbruk av kode, f.eks. ved at en eller flere av de arvede metodene ikke trenger å redefineres. Forklar med tekst og kode hvordan du vil gjøre dette. Forklar om nødvendig hvordan du må endre på **`RadioAlphabet`**-klassen for å få det til, men merk at den må fortsatt virke slik det er beskrevet i del 3.

b) Du skal gjøre **`RadioAlphabet`**-klassen (og dermed også subklassen) såkalt "observerbar", vha. følgende lyttergrensesnitt:

```
public interface RadioAlphabetListener {
   public void radioAlphabetChanged(RadioAlphabet, char);
}
```

Dersom en **`RadioAlphabet`**-instans endres, skal altså metoden **`radioAlphabetChanged`** kalles på alle lytterne, med **`RadioAlphabet`**-instansen som første argument og tegnet viss' ord ble endret/fjernet som det andre. Beskriv hvordan **`RadioAlphabet`** og evt. **`MorseAlphabet`** må endres for å implementere denne oppførselen.