Obligatorisk oppgave 4: Lege/Resept

INF1010

Frist: mandag 27. mars 2017 kl. 12:00

Versjon 1.0 (`4e2f752`)

Innhold

1 Innledning **1**
1.1 Begreper 2

2 Pasienter **2**

3 Leger og lister for leger **3**
3.1 Leger 3
3.2 Fastleger 3
3.3 Legeliste 4

4 Legemidler **4**

5 Resepter **5**

6 Klassehierarkier **6**

7 Implementasjon **6**
7.1 `toString()` 6
7.2 Hovedprogram 7

8 Oppsummering **7**

1 Innledning

I denne oppgaven skal du lage deler av et system som holder styr på leger, pasienter, resepter og legemidler. Du vil få bruk for det meste du har lært hittil i INF1010, som arv, grensesnitt, generiske klasser.

Du kommer til å få bruk for beholderne du lagde i oblig 3. Det er mulig å begynne på denne obligen før du er ferdig med oblig 3, men du trenger beholderne for å kunne kjøre hovedprogrammet og se at alt virker.

Fra tidligere år har vi erfart at mange studenter bruker svært mye tid på denne obligen. For å begrense arbeidsmengden, har vi i år gitt kraftige føringer for hvordan oppgaven skal løses. Programmet består av tre deler:

1. Et hierarki av klasser som utgjør ryggmargen i programmet.
2. Et sett av beholdere som trengs for å lagre objektene i hovedprogrammet.
3. Et kommandostyrt hovedprogram som kan lese inn data fra en fil, legge til nye objekter, skrive ut statistikk om reseptene mm.

Del 2 har du i stor grad allerede gjort i oblig 3. Del 3 vil du få ferdig kode for. Det er del 1 du skal lage i denne obligen. For at koden din skal kunne virke sammen med den ferdige koden for del 3, er det viktig at du følger den spesifikasjonen som gis senere i oppgaveteksten.

1.1 Begreper

ID. Mange av klassene i denne obligen skal kunne identifiseres ved en *ID*. Med ID menes her et unikt, ikke-negativt heltall.

Det første objektet som opprettes av en bestemt klasse skal ha `id = 0`, det andre `id = 1`, det tredje `id = 2`, osv. Merk at ID-ene ikke er unike på tvers av alle klassene, men hvert objekt av en bestemt type skal ha en id som ingen andre objekter av den samme typen har. F.eks. har ingen legemidler samme ID, og ingen pasienter har samme ID, men en pasient og et legemiddel kan ha samme ID.

Du kan se et eksempel på hvordan ID-er fordeles ved å se på en av inputfilene til hovedprogrammet. Merk at objektene med lavest id blir lest inn og opprettet først slik at ID-ene bevares når vi leser og skriver til fil.

2 Pasienter

Pasienter har et navn, et fødselsnummer (11 siffer), en gateadresse og et postnummer. Når en ny pasient registreres gis pasienten i tillegg en ID. Pasienter har også en liste over reseptene de har skrevet ut. Siden pasienten ofte vil bruke en resept kort tid etter at den er utskrevet, bruker vi en `Stabel<Resept>` til å lagre pasientens resepter.

Pasient skal ha følgende konstruktør og metoder:

```
    Pasient(String navn, long fodselsnummer, String gateadresse,
            int postnummer) { /* fyll inn */ }

    public int hentId() { /* fyll inn */ }
    public String hentNavn() { /* fyll inn */ }
    public long hentFodselsnummer() { /* fyll inn */ }
    public String hentGateadresse() { /* fyll inn */ }
    public int hentPostnummer() { /* fyll inn */ }
    public void leggTilResept(Resept resept) { /* fyll inn */ }
    public Stabel<Resept> hentReseptliste() { /* fyll inn */ }
```

3 Leger og lister for leger

3.1 Leger

En lege har et unikt navn. `Lege` skal implementere `Comparable<Lege>`, og sammenligningen skal være direkte basert på legens unike navn. For eksempel er "Dr. Oetker" mindre enn "Dr. Ueland" , så førstnevnte lege skal komme før sistnevnte lege når de ordnes i stigende rekkefølge. I tillegg har leger en liste over reseptene de har skrevet ut. Legene er ikke nødvendigvis så interessert i å se på de nyeste reseptene først, så derfor bruker vi en `Koe<Resept>` til å lagre reseptene legen har skrevet ut.

Konstruktøren i `Lege` tar kun inn en `String` med legens navn. `Lege` skal ha følgende metoder:

```
    public String hentNavn() { /* fyll inn */ }
    public int compareTo(Lege annenLege) { /* fyll inn */ }
    public void leggTilResept(Resept resept) { /* fyll inn */ }
    public Koe<Resept> hentReseptliste() { /* fyll inn */ }
```

3.2 Fastleger

Enkelte leger har en avtale med kommunen de jobber, og disse legene kalles for *fastleger*. Å ha en kommuneavtale er noe som kan gjelde andre enn leger (men i denne oppgaven blir det bare leger). Denne egenskapen skal derfor beskrives med et grensesnitt (interface). For alle som har en avtale med kommunen, skal det være mulig å hente ut et avtalenummer. Fastleger skal implementere følgende grensesnitt:

```
interface Kommuneavtale {
    public int hentAvtalenummer();
}
```

`Fastlege` arver `Lege` siden fastleger er en spesiell type leger. Konstruktøren i `Fastlege` skal ta inn `String navn` og `int avtalenummer`.

3.3 Legeliste

Vi trenger å kunne finne en lege i en beholder ved å slå opp på legens navn. Vi lager derfor en ny klasse som arver `OrdnetLenkeliste` fra oblig 3.

Fullfør følgende klasse:

```
class Legeliste extends OrdnetLenkeliste<Lege> {
    /**
     * Soeker gjennom listen etter en lege med samme navn som ‘navn‘
     * og returnerer legen (uten aa fjerne den fra listen).
     * Hvis ingen slik lege finnes, returneres ‘null‘.
     * @param   navn    navnet til legen
     * @return  legen
     */
    public Lege finnLege(String navn) {
        // fullfoer metoden
    }

    /**
     * Returnerer et String[] med navnene til alle legene i listen
     * i samme rekkefoelge som de staar i listen.
     * @return array med navn til alle legene
     */
    public String[] stringArrayMedNavn() {
        // fullfoer metoden
    }
}
```

Observér.

Vi kan lage en ikke-generisk subklasse (`Legeliste`) av en generisk klasse (`OrdnetLenkeliste<T>`) ved å sette inn en konkret type – i dette tilfellet setter vi inn `Lege` for `T`.

4 Legemidler

Et legemiddel har et navn, en ID og en pris. I tillegg må vi for alle legemidler kunne vite hvor mye virkestoff (mg) det inneholder totalt. Prisen og virkestoffet skal lagres som flyttall.

Et legemiddel er enten av type A, narkotisk, eller av type B, vanedannende, eller av type C, vanlig legemiddel. Det er stor forskjell på legemidler av disse tre typene, men i denne oppgaven skal vi bare ta hensyn til følgende krav for de forskjellige typene legemidler:

- Type A har et heltall som sier hvor sterkt narkotisk det er.
- Type B har et heltall som sier hvor vanedannende det er.
- Type C har ingen nye egenskaper (annet enn klassens navn).

Lag klassene `Legemiddel`, `LegemiddelA`, `LegemiddelB` og `LegemiddelC`. De tre sistnevnte klassene arver den førstnevnte. Konstruktøren til `LegemiddelC` skal ta inn `String navn`, `double pris` og `double virkestoff` (i den rekkefølgen). Konstruktørene til `LegemiddelA` og `LegemiddelB` skal i tillegg ta inn `int styrke`.

`Legemiddel` skal ha følgende metoder:

```
public int hentId() { /* fyll inn */ }
public String hentNavn() { /* fyll inn */  }
public double hentPris() { /* fyll inn */   }
public double hentVirkestoff() { /* fyll inn */  }
```

I tillegg har `LegemiddelA`:

```
public int hentNarkotiskStyrke() { /* fyll inn */ }
```

og `LegemiddelB`:

```
public int hentVanedannendeStyrke() { /* fyll inn */ }
```

5 Resepter

En resept har en ID. I tillegg skal en resept ha en referanse til et legemiddel, en referanse til den legen som har skrevet ut resepten, og ID-en til den pasienten som eier resepten. En resept har et antall ganger som er igjen på resepten (kalles *reit*). Hvis antall ganger igjen er 0, er resepten ugyldig.

I Norge kommer resepter i to farger: blå og hvit. Vi skal ta høyde for at det er stor forskjell på blå og hvite resepter (blant annet er utstedelsen av en blå resept forbundet med en del kontroller), men igjen skal vi gjøre en forenkling og si at bare prisen er forskjellig: Blå resepter er sterkt subsidiert, og for enkelhets skyld sier vi her at de har 75% rabatt slik at pasienten må betale 25% av prisen på legemidlet.

Skriv klassen `Resept` og dens subklasser `BlaaResept` og `HvitResept`. Konstruktørene i `BlaaResept` og `HvitResept` skal ta inn `Legemiddel legemiddel`, `Lege utskrivendeLege`, `int pasientId` og `int reit` (i den rekkefølgen).

`Resept` skal ha følgende metoder:

```
public int hentId() { /* fyll inn */ }
public Legemiddel hentLegemiddel() { /* fyll inn */ }
public Lege hentLege() { /* fyll inn */ }
public int hentPasientId() { /* fyll inn */ }
public int hentReit() { /* fyll inn */ }

/**
 * Bruker resepten én gang. Returner false om resepten er
 * oppbrukt, ellers returnerer den true.
 * @return      om resepten kunne brukes
 */
public boolean bruk() { /* fyll inn */ }
```

```java
    /**
     * Returnerer reseptens farge. Enten "blaa" eller "hvit".
     * @return      reseptens farge
     */
    abstract public String farge();

    /**
     * Returnerer prisen pasienten maa betale.
     * @return      prisen pasienten maa betale
     */
    abstract public double prisAaBetale();
```

6 Klassehierarkier

Tegn opp klassehierarkiene beskrevet over. Ta også med alle grensesnitt (interface). Ikke tegn noe mer enn navnet på grensesnittene og klassene.

7 Implementasjon

7.1 `toString()`

Noen menyer i hovedprogrammet vil skrive ut pasienter, legemidler og resepter, på steder hvor brukeren må velge blant flere objekter, og da vil `toString()` bli kalt på disse objektene. Denne metoden er definert i `Object` og vil gi en streng som ikke sier så mye annet om objektet enn hvilken klasse det er av. Du må derfor overstyre (override) `toString()` for pasienter, legemidler og resepter slik at det returneres en `String` med den nødvendige informasjonen til at du kan velge riktig pasient/legemiddel/resept i hovedprogrammet. (Nøyaktig hva som skal være med av informasjon, er det opp til deg å bestemme.) Leger er unikt identifisert ved deres navn, så hovedprogrammet vil bruke navnet deres når de skrives ut.

Under følger et eksempel på hvordan deler av en kjøring kan se ut uten og med `toString()` overstyrt i `Pasient`. Her skal brukeren velge en pasient ved å skrive inn tallet i hakeparentes.

Uten `toString()` i `Pasient`:

```
[0] Pasient@55f96302
[1] Pasient@3d4eac69
[2] Pasient@42a57993
[3] Pasient@75b84c92
[4] Pasient@6bc7c054
```

Med `toString()` i `Pasient`:

```
[0] Jens Hans Olsen (11111143521)
[1] Petrolina Swiq (24120099343)
[2] Sven Svendsen (10111224244)
[3] Juni Olsen (21049563451)
[4] Hypo Konder (00000000003)
```

7.2 Hovedprogram

Du finner hovedprogrammet på GitHub. Det er ikke et perfekt program, men det skal være vanskelig å krasje (såfremt koden din er korrekt). Det er spesielt verdt å merke seg at vi med fordel kunne flyttet mye kode ut fra hovedprogrammet og inn i klassene fra del 1, men vi har forsøkt ikke å veve disse for tett sammen.

Det er ikke viktig å forstå alle detaljene i hovedprogrammet for å kunne løse obligen. Det skal i prinsippet være tilstrekkelig å følge oppgaveteksten. Likevel vil hovedprogrammet bli gjennomgått på plenum 15. mars.

For å få obligen godkjent, må dette hovedprogrammet virke, og du bør ha testet de fleste operasjonene for å se at alt virker som det skal.

8 Oppsummering

Du skal levere tegningen av klassehierarkiet i tillegg til alle filene som trengs for at retteren din skal kunne kompilere og kjøre hovedprogrammet. Dette inkluderer:

- Grensesnitt og klasser for pasienter, leger, legemidler, resepter
- Beholderne fra oblig 3 (ikke levér testprogrammet fra oblig 3)
- Legelisten
- Hovedprogrammet med den tilhørende IO-klassen

Ikke levér zip-filer! Det går an å laste opp flere filer samtidig i Devilry.

Lykke til!
Stein Gjessing, Stein Michael og Kristian