Innledning

Eksamen INF1010 våren 2017

Totalt 100 poeng mulige + 5 bonuspoeng (i oppgave 5G)

PRAKTISK INFORMASJON

Hvis du synes noen deler av oppgaven er uklar kan du legge dine egne forutsetninger til grunn og gjøre dine egne rimelige antagelser. Gjør i så fall rede for disse forutsetningene og antagelsene. Unngå å bruke en stor del av tiden din på oppgaver som gir deg få poeng.

Faglærer prøver å besøker eksamenslokalet mellom klokka 10 og 11 for å oppklare eventuelle uklarheter og feil i oppgaveteksten.

Tillatte hjelpemidler

Eneste tillatte hjelpemiddel er læreboken, samt utdelt ekstrakapittel om tråder.

Tegning på papir

I oppgave 1A skal svaret tegnes på papir (skisseark). Instruksjon om utfylling av skisseark finner du på pulten din. Husk å notere kodenummeret og annen informasjon med én gang; du vil ikke få tid til å gjøre dette etter at eksamen er over. Det blir IKKE gitt ekstratid for å fylle ut informasjonsboksene på skisseark (engangskoder, kand.nr. o.l.). OBS: denne oppgaven er merket som type "muntlig" i oppgaveoversikten av tekniske årsaker.

Om du har behov for å tegne andre datastrukturer (tilhørende andre oppgaver enn 1A) for å vise sensor hvordan du tenker når du skriver algoritmer, så tegner du dette etter svaret på oppgave 1A, skriver tydelig på tegningen hvilken/hvilke oppgave(r) tegningen tilhører, og leverer alt som svar på oppgave 1A.

Blandede tekst/kodesvar

I noen oppgaver skal du både skrive tekst og program. Du kan da skrive teksten inn i ”programboksen” uten å passe på riktig Java-syntaks, dvs. du behøver ikke kommentere ut teksten du skriver.

Ekstraoppgave

Oppgave 5G er en ekstraoppgave. Du vil ikke tape poeng hvis du ikke løser denne, men du kan få ekstra poeng ved å gjøre det. (Eventuelle poeng over 100 vil ikke telle ved fastsettelse av karakter).

INNLEDNING

Du har søkt jobb i konsulentselskapet SykehusData og er blitt kalt inn til et intervju der du blir bedt om å vise at du er en god Java-programmerer. SykehusData legger vekt på at de ansatte er gode i objektorientert programmering, og du må derfor løse oppgavene i dette oppgavesettet for å vise dette.

Hele oppgavesettet handler om å lage deler av et datasystem for sykehus. Du skal ikke utvikle et fullstendig kjørbart program, og heller ikke et program som gjør så mye. Du skal bare lage deler av det som kunne tenkes å bli et program i et sykehus. Du vil nok også se at det kan være ting du blir bedt om å programmere som kan være nokså unaturlig og urealistisk i et virkelig sykehus. Grunnen til dette er at SykehusData ønsker at du skal vise at du er god i å programmere på forskjellige måter.

På sykehuset er det ansatte og pasienter, men oppgave 1 handler hovedsakelig om ansatte. I oppgave 2 skal du lage beholdere for pasienter, mens i oppgave 3 skal du lage avdelinger på sykehuset. I oppgave 4 skal du sette det hele sammen til et fullt sykehus. Til slutt skal du i oppgave 5 bruke tråder til å analysere sykdomsbildet hos noen pasienter.

Om klassen *Pasient*.

I mange av oppgavene skal du programmere med pasienter. Du skal bruke den definisjonen av klassen *Pasient* som du ser under. Du kan anta at alle programmene du lager i dette oppgavesettet ligger i samme pakke eller katalog som klassen *Pasient*, og programmene har dermed tilgang til å lese og skrive i alle variable i klassen.

```
class Pasient {
  final String fnr;
  final String navn;
  Pasient neste;
  final static int MAXPASPRIO = 10;
  int prioritet; // 0 <= prioritet <= MAXPASPRIO
  int sengNr = -1;
  Pasient (String navn, String fnr, int prio) {
    this.navn= navn;
    this.fnr = fnr;
    prioritet = prio;
  }
}
```

Det er naturlig at alle pasienter har en prioritet avhengig av hvor mye smerter de har og hvor syke de er. Mange pasienter vil også bli innlagt og det er derfor naturlig at de har en identifikasjon av den sengen eller den sengeplassen de ligger i, og her er dette angitt som et heltall (*sengNr*, seng nummer). De andre variablene og konstantene i klassen vil bli forklart etter hvert.

NB: ved kopiering og liming av kode vil kodeboksen sette inn ekstra blanke linjer mellom hver kodelinje. Vi er klar over dette problemet som dessverre skyldes tekniske årsaker utenfor vår kontroll. Du vil ikke bli trukket for eventuelle formatteringsproblemer relatert til dette.

1(a) Oppgave 1A

I denne oppgaven skal du bruke digital håndtegning. Bruk eget skisseark (utdelt). Se instruksjon for utfylling av skisseark på pult.

Tegn opp klassehierarkiet for ansatte på sykehuset. Ikke ta med variable, konstanter eller metoder.
Hvis du er i tvil om noe kan det hende du får noen tips ved å lese videre om de enkelte ansatte.

Maks poeng: 2

1(b) Oppgave 1B

Skriv i Java alle deler av dette klassehierarkiet som har med leger å gjøre (og som følger beskrivelsen i oppgave 1A og innledningen til oppgave 1B). Sagt på en annen måte: Du skal skrive alle delene av dette klassehierarkiet unntatt det som har med sykepleiere å gjøre.

Skriv ditt svar her...

```
1
```

Maks poeng: 9

2(a) Oppgave 2A

Skriv klassen *PasientAdm*.

Skriv ditt svar her...

```
1
```

Maks poeng: 4

2(b) Oppgave 2B

Skriv klassen *PasientPrio*.

Skriv ditt svar her...

```
1
```

Maks poeng: 16

2(c) Oppgave 2C

Skriv klassen *PasientTabell*.

Skriv ditt svar her...

```
1
```

Maks poeng: 12

3(a)

Oppgave 3A

Skriv klassen *Avdeling.*

Skriv ditt svar her...

```
1
```

Maks poeng: 10

3(b)

Oppgave 3B

Skriv de to klassene *Akutten* og *Sengepost.*

Skriv ditt svar her...

```
1
```

Maks poeng: 10

4(a) Oppgave 4A

Skriv de tre metodene som klassen *Sykehus* skal inneholde.

Skriv ditt svar her...

```
1
```

Maks poeng: 5

4(b) Oppgave 4B

Skriv klassen *Sykehus* med to avdelinger, en akuttavdeling og en sengepost med 100 senger og en *main*-metode som:

- Oppretter to pasienter og skriver dem inn på (setter dem inn i) akutt-avdelingen.
- Overfører en av pasientene fra akutt-avdelingen til sengeposten
- Skriver den andre pasienten ut fra (henter pasienten ut fra) akutt-avdelingen.

Skriv ditt svar her...

```
1
```

Maks poeng: 7

5(a) Oppgave 5A

1. Forklar kort hvilke forandringer du må gjøre i programmene du laget som svar på oppgavene 2, 3 og 4 når klassen *Pasient* blir abstrakt?
2. Skriv den abstrakte superklassen *Pasient* og de to subklassene *KvinneligPasient* og *MannligPasient*.
3. Beskriv hvordan programmet ditt i resten av oppgave 5 skal kalle de fire statiske metodene i *Sykehus* på en mest mulig objektorientert måte. Skriv gjerne en eller flere korte kodebiter som viser dette og som du kan bruke senere i oppgave 5.

NB: ved kopiering og liming av kode vil kodeboksen sette inn ekstra blanke linjer mellom hver kodelinje. Vi er klar over dette problemet som dessverre skyldes tekniske årsaker utenfor vår kontroll. Du vil ikke bli trukket for eventuelle formatteringsproblemer relatert til dette.

Skriv ditt svar her...

Maks poeng: 6

5(b) Oppgave 5B

Du skal ikke modifisere klassen *PasientPrio* i dette oppgavesettet, og i ditt program skal sil-trådene bruke denne klassen slik den er. Men hva mener du burde vært gjort med klassen *PasientPrio* siden den skal brukes av mange tråder samtidig. Begrunn svaret ditt.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 2

5(c) Oppgave 5C

Skriv tråd-klassen *Sil*.

Parametrene til konstruktøren skal inneholde referanser til *PasientPrio*-objektet og *AnalyseBuffer*-objektet. En tråd terminerer når den ikke får returnert et pasient-objekt fra *PasientPrio*-objektet.

Skriv ditt svar her...

```
1
```

Maks poeng: 5

5(d) Oppgave 5D

Skriv klassen *AnalyseBuffer*. Rekkefølgen på innseting og uttak bør være først-inn-først-ut, men ellers kan du velge hva slags datastruktur du vil bruke, og det er lov å hente noe fra Java-biblioteket.

Skriv ditt svar her...

```
1
```

Maks poeng: 5

5(e) Oppgave 5E

Skriv tråd-klassen *Analyse*. Ikke tenk på at trådene skal terminere.

Skriv ditt svar her...

```
1
```

Maks poeng: 2

5(f) Oppgave 5F

Skriv en metode kalt *utforAnalyse* (utfør analyse) som har to parametre. Den ene er en referanse til et *PasientPrio*-objekt, den andre er antall analysetråder som skal opprettes.

Metoden skal:

- Opprette et *AnalyseBuffer*,
- opprette og starte *MAXPASPRIO* + *1* tråder av klassen *Sil* og
- opprette og starte tråder av klassen *Analyse*

Husk at i denne oppgaven skal du ikke prøve å terminere analyse-trådene på en fornuftig måte

Skriv ditt svar her...

```
1
```

Maks poeng: 5

5(g) Ekstraoppgave 5G

NB! Dette er en **ekstraoppgave**. Hvis du ikke løser denne oppgaven kan du fortsatt få 100 poeng og full pott.

Forklar og begrunn meget kort hvordan dette programmet kan modifiseres slik at alle trådene allikevel terminerer på en fornuftig måte?

Ta en kopi av programkoden din fra besvarelsen av oppgavene 5C – 5F og modifiser den slik at alle trådene terminerer fornuftig og *utforAnalyse*-metoden først terminerer når alle trådene også har terminert.

For at sensor skal se hvilke forandringer du har gjort på programmet ditt skal du skrive en en-bokstavs kommentar (F for forandring) på slutten av alle linjene du modifiserer eller legger inn, for eksempel slik : *// F*

NB: ved kopiering og liming av kode vil kodeboksen sette inn ekstra blanke linjer mellom hver kodelinje. Vi er klar over dette problemet som dessverre skyldes tekniske årsaker utenfor vår kontroll. Du vil ikke bli trukket for eventuelle formatteringsproblemer relatert til dette.

Skriv ditt svar her...

```
1
```

Maks poeng: 5

Question 1.a
Attached

Exercise 1 (11 points in total)

Introduction to exercise 1A

There are two kinds of employees at the hospital: doctors and nurses. Some doctors are consultants (chief physician). Some consultants and some nurses have special training so that they have an additional role as administrators. The program must be able to deal with this role in the same way independently of whether an employee is a consultant or a nurse.

Introduction to exercise 1B

All employees have an employee identifier (a String) and a name (also a String). Both attributes are constants and are specified as an object is being created. It should not be possible to create any objects of this class. Doctors have an additional doctor identification number (an integer). All consultants have a specialty type (String). All of these attributes are constants as well and are specified when creating objects of these classes. Do not write methods for reading the constants. Those consultants and nurses who are also administrators have a department code (a String), and we must be able to retrieve the department code of any administrator. This code is also specified at object creation time.

Question 1.a
Attached

Oppgave 1 (totalt 11 poeng)

Innledning til oppgave 1A

Det er to typer ansatte på sykehuset: Leger og sykepleiere. Noen leger er overleger. Noen overleger og noen sykepleiere har tatt en egen utdanning som gjør at de i tillegg har den ekstra rollen at de er administratorer. Dette er en rolle som programmet skal kunne behandle på samme måte uavhengig om det er overleger eller sykepleiere.

Innledning til oppgave 1B

Alle ansatte har en ansattidentifikasjon (en String) og et navn (også en String). Begge deler er konstanter og oppgis i det et objekt opprettes. Det skal ikke kunne lages objekter av denne klassen. En lege har i tillegg et legeNummer (et heltall). Alle overleger har en spesialiseringstype (String). Alle disse egenskapene er også konstanter og skal oppgis i det det lages objekter av disse klassene. Ikke lag metoder for å lese av konstantene. De overlegene og sykepleierne som i tillegg er administratorer har en ansvarskode (en String), og for alle administratorer skal vi kunne finne ut deres ansvarskode. Også denne koden skal oppgis når det opprettes objekter av disse klassene.

Question 1.b
Attached

Exercise 1 (11 points in total)

Introduction to exercise 1A

There are two kinds of employees at the hospital: doctors and nurses. Some doctors are consultants (chief physician). Some consultants and some nurses have special training so that they have an additional role as administrators. The program must be able to deal with this role in the same way independently of whether an employee is a consultant or a nurse.

Introduction to exercise 1B

All employees have an employee identifier (a String) and a name (also a String). Both attributes are constants and are specified as an object is being created. It should not be possible to create any objects of this class. Doctors have an additional doctor identification number (an integer). All consultants have a specialty type (String). All of these attributes are constants as well and are specified when creating objects of these classes. Do not write methods for reading the constants. Those consultants and nurses who are also administrators have a department code (a String), and we must be able to retrieve the department code of any administrator. This code is also specified at object creation time.

Question 1.b
Attached

Oppgave 1 (totalt 11 poeng)

Innledning til oppgave 1A

Det er to typer ansatte på sykehuset: Leger og sykepleiere. Noen leger er overleger. Noen overleger og noen sykepleiere har tatt en egen utdanning som gjør at de i tillegg har den ekstra rollen at de er administratorer. Dette er en rolle som programmet skal kunne behandle på samme måte uavhengig om det er overleger eller sykepleiere.

Innledning til oppgave 1B

Alle ansatte har en ansattidentifikasjon (en String) og et navn (også en String). Begge deler er konstanter og oppgis i det et objekt opprettes. Det skal ikke kunne lages objekter av denne klassen. En lege har i tillegg et legeNummer (et heltall). Alle overleger har en spesialiseringstype (String). Alle disse egenskapene er også konstanter og skal oppgis i det det lages objekter av disse klassene. Ikke lag metoder for å lese av konstantene. De overlegene og sykepleierne som i tillegg er administratorer har en ansvarskode (en String), og for alle administratorer skal vi kunne finne ut deres ansvarskode. Også denne koden skal oppgis når det opprettes objekter av disse klassene.

Question 2.a
Attached

Introduction to exercise 2. (29 points in total)

In this exercise, you will create an abstract superclass with two subclasses. These classes should be able to administrate *Patient* objects. You will need to make use of these in exercise 3, so we encourage you to read exercise 3 before solving exercise 2.

The abstract superclass shall be named *PatientAdm* (short for patient administration) and is meant to administrate patients who are hospitalized. The class should have three methods: *void insertPatient(Patient p), Patient retrieve(Patient p)* and *Patient retrieve(int i)*, where *i* is a patient priority. Both *retrieve* methods must remove the patient from the container.

Question 2.a
Attached

Innledning til oppgave 2. (totalt 32 poeng)

Du skal i denne oppgaven lage en abstrakt superklasse med to subklasser. Disse klassene skal kunne administrere *Pasient*-objekter. Du vil få bruk for disse i oppgave 3, så les gjerne oppgave 3 før du begynner å besvare oppgave 2.

Den abstrakte superklassen skal hete *PasientAdm* (forkortelse for pasientadministrasjon) og brukes til å administrere pasienter som er på sykehuset. Klassen skal ha tre metoder: *void settInnPasient(Pasient p)*, *Pasient hentUt(Pasient p)* og *Pasient hentUt(int i)*, der *i* er en pasientprioritet. Begge *hentUt*-metodene skal fjerne pasienten fra beholderen.

Question 2.b
Attached

Introduction to exercise 2B

The class *PatientPrio* (patient priority) shall be a subclass of *PatientAdm*. Using the *PatientPrio* class, the program shall be able to insert patients according to priority, observing a First-In-First-Out (FIFO) ordering within the same priority. The motivation for this class is that when patients arrive at the hospital, a nurse assigns them a priority depending on how urgently the patients need treatment. To facilitate this, the program shall contain one list per priority.

You must write a linked lists implementation yourself that adheres to this FIFO principle within the same priority by using the *next*-reference that already exists within the *Patient* objects. *PatientPrio* must contain two arrays of length *MAXPATPRIO + 1* and of type *Patient*, where the array index denotes priority. One array contains the heads of the lists, the other the tails of the lists, see the illustration for a *MAXPATPRIO* value of 3 below. Since we also employ priority 0 the program will thusly contain *MAXPATPRIO + 1* lists. When inserting a patient, the program must use the patient's priority to find the appropriate list to insert her into. The method *retrieve(i)* retrieves records from the list of index *i* according to the FIFO-principle. The method *retrieve(p)* must use the patient's priority to select the appropriate list, and retrieve the object that *p* refers to from the list. This method must also return a reference to the retrieved patient, or *null* in case the specified patient is not found in the list.

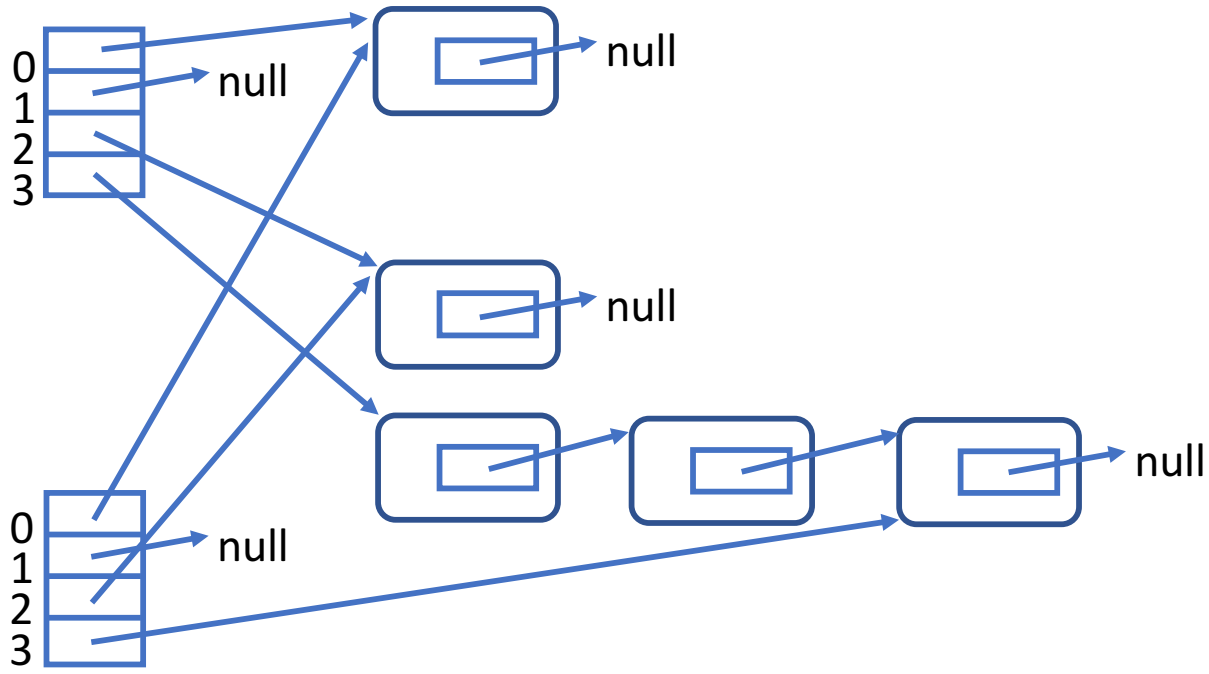


Illustration with two arrays of length 4 and a few patient objects that are linked together.

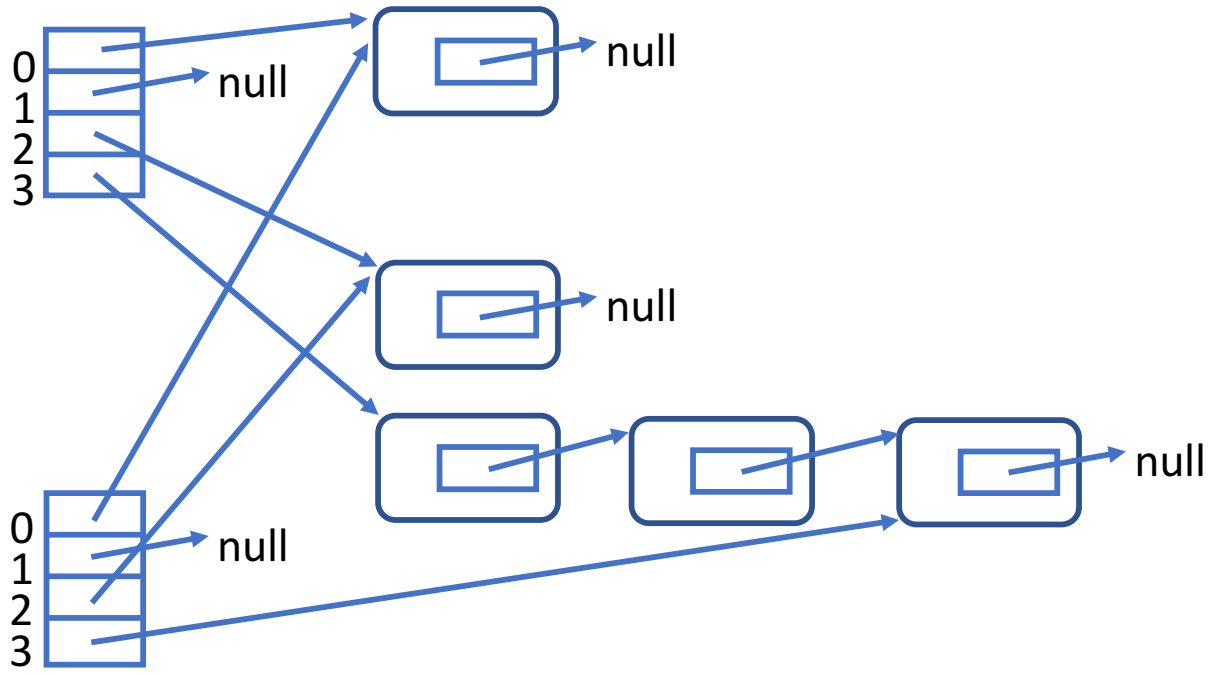
(Admittedly, we probably wouldn't use two arrays in this manner to manage these queues in a real program. We require you to do it this way so as to allow you to demonstrate your ability to program linked lists).

Question 2.b
Attached

Innledning til oppgave 2B

Klassen *PasientPrio* (pasientprioritet) skal være en subklasse av *PasientAdm*. I klassen *PasientPrio* skal programmet kunne sett inn pasienter etter prioritet, og det skal være først-inn-først-ut (first-in-first-out, FIFO) rekkefølge innen samme prioritet. Motivasjonen bak denne klassen er at når pasienter kommer til sykehuset vil en sykepleier gi pasienten en prioritet avhengig av hvor fort pasienten må behandles. Derfor skal programmet inneholde en liste per prioritet.

Du skal selv skrive lenkete lister som implementerer dette FIFO-prinsippet innen samme prioritet ved å bruke neste-referansen som alt finnes i Pasient-objektene. *PasientPrio* skal inneholden to arrayer av lengde *MAXPASPRIO + 1* og av type *Pasient*, der indeksen angir prioritet. Den ene arrayen inneholder pekere til startene av listene, den andre arrayen inneholder pekere til objektene som ligger bakerst i listene, se figur med *MAXPASPRIO* lik 3 nedenfor. Siden vi også bruker prioritet 0 vil programmet følgelig inneholde *MAXPASPRIO + 1* antall lister. Når en pasient settes inn skal programmet bruke pasientens prioritet til å finne riktig liste å sette inn i. Metoden *hentUt(i)* henter ut fra listen med indeks *i* etter FIFO-prinsipp. Metoden *hentUt(p)* skal bruke pasientens prioritet til å finne riktig liste, og hente objektet *p* peker på ut av listen. Den siste metoden skal også returnere en referanse til pasienten som ble tatt ut, men om denne pasienten ikke blir funnet i listen skal metodene returnere *null*.



Figur med to arrayer av lengde 4 og noen pasientobjekter som er lenket sammen.

(I parentes bemerkes det at vi nok ikke ville bruke to slike arrayer på denne måten i et virkelig program, men det gjøres her for at du skal vise at du behersker å programmere lenkede lister)

Question 2.c
Attached

Introduction to exercise 2C

The *PatientTable* class shall be a subclass of *PatientAdm*. Objects of the *PatientTable* class shall contain an array of type *Patient*. The length of this array is specified when creating a new object. The motivation for this class is that every index in the array represents a bed or bed location within the hospital.
When an element in the array is *null*, that bed is considered vacant.
When an element contains a (non-*null*) reference to a patient, then that patient is occupying the location.
When inserting patients, priority shall not be taken into consideration. However, in order to ensure that all locations may be taken into use, you may not start to search for a vacant location starting at index 0, but rather starting where the last patient was inserted (upon the previous call to *insertPatient*).
Programming this correctly is an important part of this exercise.
When a patient has been assigned a location in the array, the identifier (index) of this location must be assigned to the variable *bedNo* in the patient object.

Question 2.c
Attached

Innledning til oppgave 2C

Klassen *PasientTabell* skal være en subklasse av *PasientAdm*. Objekter av klassen *PasientTabell* skal inneholde en array av typen *Pasient*. Lengden på denne arrayen oppgis når det opprettes et nytt objekt.
Motivasjonen bak klassen er at hver indeks i arrayen kan angi nummeret på en seng eller en sengeplass på sykehuset.
Når en referanse i arrayen er *null*, er denne sengen eller sengeplassen ledig.
Når referansen ikke er *null* peker den på pasienten som ligger på denne plassen.
Når du setter inne en pasient spiller ikke prioriteten noen rolle. Men for at alle senger eller sengeplasser skal brukes, skal du ikke starte leting etter ledig sengeplass fra 0, men fra der siste pasient ble satt inn (ved forrige kall på *settInnPasient*).
At du programmerer dette riktig er en viktig del av denne oppgaven.
Når en pasient har fått en plass i arrayen skal nummeret på denne plassen tilordnes variabelen *sengNr* i pasient-objektet.
Metoden *hentUt(i)* skal finne og ta ut en (vilkårlig) pasient med prioritet lik i. Let gjerne fra indeks 0 og oppover. Om det ikke finnes noen pasient med denne prioriteten returneres null. Metoden *hentUt(p)* skal bruke pasientens *sengNr* til å fjerne pasienten fra beholderen. Når en pasient tas ut settes *sengNr* til -1.

Question 3.a
Attached

Exercise 3 (20 points in total)
The program you write to solve this exercise may be short compared to the amount of points at stake.
Therefore, it is important that you take your time considering how to write a good solution.

In this exercise you will start off by defining a *Ward* class that will be the superclass of all hospital wards (departments). Then you will define two subclasses, *EmergencyDept* and *InpatientWard*.
The manner in which all wards receive and retrieve patients must be defined in the *Ward* superclass. Any ward must be able to administrate the patients that are present in the ward, therefore all wards must contain an object which is a subclass of *PatientAdm*.

The *Ward* class shall contain three methods:

- *void insert(Patient p)*
- *Patient retrieve(Patient p)*
- *Patient retrieve(int i)*, where *i* is a priority.

These methods must not be abstract, but rather contain full implementations which should not be overridable in subclasses.

The classes *EmergencyDept* and *InpatientWard* shall both be subclasses of *Ward*.

In the *EmergencyDept* class all patients will be inserted and removed according to their priority. Use the *PatientPrio* class to administrate the patients of this ward.

In the *InpatientWard* class all patients will be inserted and removed with respect to the available beds on the ward. An inpatient ward is a ward with a given number of beds, specified when calling the constructor of the class. Use the *PatientTable* class to implement this. (It is somewhat artificial that patients can be retrieved based on priority within this ward, but that is how it works for the purpose of this exercise).

Question 3.a
Attached

Oppgave 3 (totalt 20 poeng)
I denne oppgaven kan programmet du lager være kort i forhold til antall poeng.
Det er derfor viktig at du bruker tid på å skjønne hvordan en god løsning skal programmeres.

Du skal i denne oppgaven først definere en klasse *Avdeling* som skal være superklassen til alle avdelingene på sykehuset. Deretter skal du definere to subklasser, *Akutten* og *Sengepost*.
Måten alle avdelinger tar imot og tar ut pasienter skal defineres i superklassen *Avdeling*. Enhver avdeling må kunne administrere de pasientene som er på denne avdelingen, derfor må alle avdelinger inneholde et objekt som er en subklasse av *PasientAdm*.

Klassen *Avdeling* skal inneholde tre metoder:

- *void settInn(Pasient p)*
- *Pasient hentUt(Pasient p)*
- *Pasient hentUt(int i)*, der *i* er en prioritet.

De tre metodene skal ikke være abstrakte, men derimot inneholde full kode som ikke skal kunne redefineres i subklasser.

Klassene *Akutten* og *Sengepost* skal begge være subklasser til *Avdeling*.

I klassen *Akutten* skal alle pasienter tas imot og settes inn basert på prioritet. Bruk klassen *PasientPrio* til å administrere pasientene på denne avdelingen.

I klassen *Sengepost* skal alle pasienter tas imot og settes inn basert på ledige senger eller sengeplasser. En sengepost er en avdeling med et antall senger som angis som en konstruktør til avdelingen. Bruk klassen *PasientTabell* til dette. (Det er litt kunstig at pasienter kan bli tatt ut fra en sengepost basert på prioritet, men det er slik det er i denne oppgaven).

Question 3.b
Attached

Exercise 3 (20 points in total)
The program you write to solve this exercise may be short compared to the amount of points at stake.
Therefore, it is important that you take your time considering how to write a good solution.

In this exercise you will start off by defining a *Ward* class that will be the superclass of all hospital wards (departments). Then you will define two subclasses, *EmergencyDept* and *InpatientWard*.
The manner in which all wards receive and retrieve patients must be defined in the *Ward* superclass. Any ward must be able to administrate the patients that are present in the ward, therefore all wards must contain an object which is a subclass of *PatientAdm*.

The *Ward* class shall contain three methods:

- *void insert(Patient p)*
- *Patient retrieve(Patient p)*
- *Patient retrieve(int i)*, where *i* is a priority.

These methods must not be abstract, but rather contain full implementations which should not be overridable in subclasses.

The classes *EmergencyDept* and *InpatientWard* shall both be subclasses of *Ward*.

In the *EmergencyDept* class all patients will be inserted and removed according to their priority. Use the *PatientPrio* class to administrate the patients of this ward.

In the *InpatientWard* class all patients will be inserted and removed with respect to the available beds on the ward. An inpatient ward is a ward with a given number of beds, specified when calling the constructor of the class. Use the *PatientTable* class to implement this. (It is somewhat artificial that patients can be retrieved based on priority within this ward, but that is how it works for the purpose of this exercise).

Question 3.b
Attached

Oppgave 3 (totalt 20 poeng)
I denne oppgaven kan programmet du lager være kort i forhold til antall poeng.
Det er derfor viktig at du bruker tid på å skjønne hvordan en god løsning skal programmeres.

Du skal i denne oppgaven først definere en klasse *Avdeling* som skal være superklassen til alle avdelingene på sykehuset. Deretter skal du definere to subklasser, *Akutten* og *Sengepost*.
Måten alle avdelinger tar imot og tar ut pasienter skal defineres i superklassen *Avdeling*. Enhver avdeling må kunne administrere de pasientene som er på denne avdelingen, derfor må alle avdelinger inneholde et objekt som er en subklasse av *PasientAdm*.

Klassen *Avdeling* skal inneholde tre metoder:

- *void settInn(Pasient p)*
- *Pasient hentUt(Pasient p)*
- *Pasient hentUt(int i)*, der *i* er en prioritet.

De tre metodene skal ikke være abstrakte, men derimot inneholde full kode som ikke skal kunne redefineres i subklasser.

Klassene *Akutten* og *Sengepost* skal begge være subklasser til *Avdeling*.

I klassen *Akutten* skal alle pasienter tas imot og settes inn basert på prioritet. Bruk klassen *PasientPrio* til å administrere pasientene på denne avdelingen.

I klassen *Sengepost* skal alle pasienter tas imot og settes inn basert på ledige senger eller sengeplasser. En sengepost er en avdeling med et antall senger som angis som en konstruktør til avdelingen. Bruk klassen *PasientTabell* til dette. (Det er litt kunstig at pasienter kan bli tatt ut fra en sengepost basert på prioritet, men det er slik det er i denne oppgaven).

Question 4.a
Attached

Exercise 4 (12 points in total)

In this exercise, you will define a *Hospital* class with two wards: an emergency department and an inpatient ward. You will write a *main* method such that an emergency department and an inpatient ward are created, before creating 2 patients who are then inserted into the emergency department. Then, a patient will be transferred from the emergency department to the inpatient ward, and finally another patient will be discharged from the emergency department.

The *Hospital* class shall contain three methods so that: one admits a patient to a ward (i.e. inserts the patient into that ward), another method transfers a patient from one ward to another and a third method that discharges a patient from a ward (removes a patient from that ward), respectively.
All of these methods receive a reference to the patient in question as a parameter. The admission and discharge methods must also receive a reference to the ward in question, whilst the transferal method must receive references to both wards concerned.

Question 4.a
Attached

Oppgave 4 (totalt 12 poeng)

I denne oppgaven skal du skrive en klasse *Sykehus* med to avdelinger, en akutt-avdeling og en sengepost. Til slutt skal du skrive en *main*-metode slik at det opprettes en akutt-avdeling og en sengepost-avdeling, videre skal det opprettes 2 pasienter som settes inn på akutten. Deretter skal en pasient overføres fra akutten til sengeposten, og til slutt skal en pasient skrives ut fra akutten.

Klassen *Sykehus* skal inneholde tre metoder som hhv. skriver en pasient inn på en avdeling (dvs setter pasienten inn på denne avdelingen), overfører en pasient fra en avdeling til en annen og skriver en pasient ut fra en avdeling (tar pasienten ut av avdelingen). Alle metodene skal ha en referanse til en pasient som parameter. Skriv inn og skriv ut skal ha en referanse til den aktuelle avdeling, men overfør skal ha to parametre i tillegg til pasienten (referanser til de to aktuelle avdelingene)

Question 4.b
Attached

Exercise 4 (12 points in total)

In this exercise, you will define a *Hospital* class with two wards: an emergency department and an inpatient ward. You will write a *main* method such that an emergency department and an inpatient ward are created, before creating 2 patients who are then inserted into the emergency department. Then, a patient will be transferred from the emergency department to the inpatient ward, and finally another patient will be discharged from the emergency department.

The *Hospital* class shall contain three methods so that: one admits a patient to a ward (i.e. inserts the patient into that ward), another method transfers a patient from one ward to another and a third method that discharges a patient from a ward (removes a patient from that ward), respectively.
All of these methods receive a reference to the patient in question as a parameter. The admission and discharge methods must also receive a reference to the ward in question, whilst the transferal method must receive references to both wards concerned.

Question 4.b
Attached

Oppgave 4 (totalt 12 poeng)

I denne oppgaven skal du skrive en klasse *Sykehus* med to avdelinger, en akutt-avdeling og en sengepost. Til slutt skal du skrive en *main*-metode slik at det opprettes en akutt-avdeling og en sengepost-avdeling, videre skal det opprettes 2 pasienter som settes inn på akutten. Deretter skal en pasient overføres fra akutten til sengeposten, og til slutt skal en pasient skrives ut fra akutten.

Klassen *Sykehus* skal inneholde tre metoder som hhv. skriver en pasient inn på en avdeling (dvs setter pasienten inn på denne avdelingen), overfører en pasient fra en avdeling til en annen og skriver en pasient ut fra en avdeling (tar pasienten ut av avdelingen). Alle metodene skal ha en referanse til en pasient som parameter. Skriv inn og skriv ut skal ha en referanse til den aktuelle avdeling, men overfør skal ha to parametre i tillegg til pasienten (referanser til de to aktuelle avdelingene)

Question 5.a
Attached

Exercise 5. (25 points in total + 5 bonus points)

Female and male patients, and processing of patient data using threads.

The first part of this exercise revolves around turning the class *Patient* into an abstract class, subsequently defining two subclasses, *FemalePatient* and *MalePatient*. Then you will process patient data using multiple threads. You will write two thread classes, one class to filter out (*class Sieve*) patients that are in the risk group for contracting a disease, and a class (*class Analysis*) that further analyses these patients.

There shall be one sieve (filter) thread for each priority, that is *MAXPATPRIO + 1* sieve threads. Sieve thread no. *i* must read data from an object of the *PatientPrio* class by calling the *retrieve(i)*method. You must use the *PatientPrio* class even if you have not solved exercise 2.
For every patient retrieved, a static method in the *Hospital* class should be called. These methods return *true* if this patient is in the risk group and thus requires further attention, or *false* in the inverse case. We will assume that these are preexisting methods, written a long time ago and that you may not modify. For males, the method is called *public static boolean inRiskGroupMale(Patient p)*, for women the method is *public static boolean inRiskGroupFemale(Patient p)*. All patients found to be in the risk group are placed in a buffer of the class *AnalysisBuffer*. The sieve threads terminate when the *retrieve()* methods return *null*.

The analysis threads must retrieve patients from the analysis buffer. For each retrieved patient, a static method in the *Hospital* class should be called. In the male case, this method is *public static void potentiallyInfectedMale(Patient p)*, in the female case it is called *public static void potentiallyInfectedFemale(Patient p)*. As above, these methods were written a long time ago and can not be modified, only called from your program. Further processing of these risk patients takes place within these methods. Consequently, the analysis threads do not need to take any further action after these methods return and may retrieve the next patient at that time.

Introduction to exercise 5A (Exercise 5A does not involve any threads).

Copy the code for the *Patient* class from the main introduction section and modify it so that *Patient* becomes an abstract class. Further, write two subclasses, *FemalePatient* and *MalePatient*.
In later parts of exercise 5 you will need to call the four static methods in *Hospital* to process patient data. Your program needs to call different methods for men and women, and you may now assume that you will be dealing with objects of the classes *FemalePatient* and *MalePatient*.
It may be wise to understand (and perhaps even solve) the rest of exercise 5 before answering exercise 5A.

Question 5.a
Attached

Oppgave 5 (totalt 25 poeng + 5 bonuspoeng)

Kvinnelige og Mannlige pasienter og behandling av pasientdata ved hjelp av tråder.

Starten av denne oppgaven dreier seg om å lage klassen *Pasient* abstrakt, og deretter lage to subklasser av denne klassen, kalt *KvinneligPasient* og *MannligPasient*. Deretter skal du behandle pasientdata ved hjelp av tråder. Du skal lage to trådklasser, en klasse for å sile ut (*class Sil*) pasienter som er i faresonen for å få en sykdom, og en klasse (*class Analyse*) som analyserer disse pasientene.

Det skal være en sil-tråd for hver prioritet, altså *MAXPASPRIO + 1* sil-tråder. Sil-tråd nr *i* skal lese data fra et objekt av klassen *PasientPrio* ved å kalle metoden *hentUt(i)*. Du skal bruke klassen *PasientPrio* selv om du ikke har svart på oppgave 2.
For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus* som returnerer *true* hvis dette er en pasient som er i faresonen og derfor skal analyseres, *false* hvis ikke. Dette er metoder som allerede finnes, som ble laget for lenge siden og som du ikke kan forandre. For menn heter metoden *public static boolean iFaresonenMann(Pasient p)*, for kvinner heter metoden *public static boolean iFaresonenKvinne(Pasient p)*.
Alle pasienter som er i faresonen skal legges inn i et buffer av klassen *AnalyseBuffer*.
Sil-trådene terminerer når *hentUt*-metodene returnerer *null*.

Analyse-trådene skal hente pasienter fra analysebufferet. For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus*. For menn skal metoden *public static void kanHaSykdomMann(Pasient p)* kalles, for kvinner skal metoden *public static void kanHaSykdomKvinne(Pasient p)* kalles. Dette er også metoder som ble laget for lenge siden, og som du ikke kan gjøre noe med, bare kalle fra ditt program. Videre behandling av de pasientene som kan ha sykdommen skjer i disse metodene. Det betyr at når disse metodene returnerer skal analyse-trådene ikke gjøre noe mer med denne pasienten, men bare hente ut en ny pasient fra analysebufferet.

Innledning til oppgavene 5A (Oppgave 5A har ikke noe med tråder å gjøre).

Du skal kopiere klassen *Pasient* fra oppgavesettets innledning og modifisere den slik at den blir abstrakt. Deretter skal du lage to subklasser *KvinneligPasient* og *MannligPasient*. I de to trådklassene som du skal skrive senere i oppgave 5 skal du kalle de fire statiske metodene i *Sykehus* for å behandle pasientdata. Programmet må kalle forskjellige metoder for kvinner og menn og nå kan du anta at du har med objekter av klassene *KvinneligPasient* og *ManneligPasient* å gjøre.
Det kan derfor være lurt å forstå (og kanskje til å med løse) resten av oppgave 5 før du besvarer oppgave 5A.

Question 5.b
Attached

Exercise 5. (25 points in total + 5 bonus points)

Female and male patients, and processing of patient data using threads.

The first part of this exercise revolves around turning the class *Patient* into an abstract class, subsequently defining two subclasses, *FemalePatient* and *MalePatient*. Then you will process patient data using multiple threads. You will write two thread classes, one class to filter out (*class Sieve*) patients that are in the risk group for contracting a disease, and a class (*class Analysis*) that further analyses these patients.

There shall be one sieve (filter) thread for each priority, that is *MAXPATPRIO + 1* sieve threads. Sieve thread no. *i* must read data from an object of the *PatientPrio* class by calling the *retrieve(i)*method. You must use the *PatientPrio* class even if you have not solved exercise 2.
For every patient retrieved, a static method in the *Hospital* class should be called. These methods return *true* if this patient is in the risk group and thus requires further attention, or *false* in the inverse case. We will assume that these are preexisting methods, written a long time ago and that you may not modify. For males, the method is called *public static boolean inRiskGroupMale(Patient p)*, for women the method is *public static boolean inRiskGroupFemale(Patient p)*. All patients found to be in the risk group are placed in a buffer of the class *AnalysisBuffer*. The sieve threads terminate when the *retrieve()* methods return *null*.

The analysis threads must retrieve patients from the analysis buffer. For each retrieved patient, a static method in the *Hospital* class should be called. In the male case, this method is *public static void potentiallyInfectedMale(Patient p)*, in the female case it is called *public static void potentiallyInfectedFemale(Patient p)*. As above, these methods were written a long time ago and can not be modified, only called from your program. Further processing of these risk patients takes place within these methods. Consequently, the analysis threads do not need to take any further action after these methods return and may retrieve the next patient at that time.

Introduction to exercise 5A (Exercise 5A does not involve any threads).

Copy the code for the *Patient* class from the main introduction section and modify it so that *Patient* becomes an abstract class. Further, write two subclasses, *FemalePatient* and *MalePatient*.
In later parts of exercise 5 you will need to call the four static methods in *Hospital* to process patient data. Your program needs to call different methods for men and women, and you may now assume that you will be dealing with objects of the classes *FemalePatient* and *MalePatient*.
It may be wise to understand (and perhaps even solve) the rest of exercise 5 before answering exercise 5A.

Question 5.b
Attached

Oppgave 5 (totalt 25 poeng + 5 bonuspoeng)

Kvinnelige og Mannlige pasienter og behandling av pasientdata ved hjelp av tråder.

Starten av denne oppgaven dreier seg om å lage klassen *Pasient* abstrakt, og deretter lage to subklasser av denne klassen, kalt *KvinneligPasient* og *MannligPasient*. Deretter skal du behandle pasientdata ved hjelp av tråder. Du skal lage to trådklasser, en klasse for å sile ut (*class Sil*) pasienter som er i faresonen for å få en sykdom, og en klasse (*class Analyse*) som analyserer disse pasientene.

Det skal være en sil-tråd for hver prioritet, altså *MAXPASPRIO + 1* sil-tråder. Sil-tråd nr *i* skal lese data fra et objekt av klassen *PasientPrio* ved å kalle metoden *hentUt(i)*. Du skal bruke klassen *PasientPrio* selv om du ikke har svart på oppgave 2.
For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus* som returnerer *true* hvis dette er en pasient som er i faresonen og derfor skal analyseres, *false* hvis ikke. Dette er metoder som allerede finnes, som ble laget for lenge siden og som du ikke kan forandre. For menn heter metoden *public static boolean iFaresonenMann(Pasient p)*, for kvinner heter metoden *public static boolean iFaresonenKvinne(Pasient p)*.
Alle pasienter som er i faresonen skal legges inn i et buffer av klassen *AnalyseBuffer*.
Sil-trådene terminerer når *hentUt*-metodene returnerer *null*.

Analyse-trådene skal hente pasienter fra analysebufferet. For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus*. For menn skal metoden *public static void kanHaSykdomMann(Pasient p)* kalles, for kvinner skal metoden *public static void kanHaSykdomKvinne(Pasient p)* kalles. Dette er også metoder som ble laget for lenge siden, og som du ikke kan gjøre noe med, bare kalle fra ditt program. Videre behandling av de pasientene som kan ha sykdommen skjer i disse metodene. Det betyr at når disse metodene returnerer skal analyse-trådene ikke gjøre noe mer med denne pasienten, men bare hente ut en ny pasient fra analysebufferet.

Innledning til oppgavene 5A (Oppgave 5A har ikke noe med tråder å gjøre).

Du skal kopiere klassen *Pasient* fra oppgavesettets innledning og modifisere den slik at den blir abstrakt. Deretter skal du lage to subklasser *KvinneligPasient* og *MannligPasient*. I de to trådklassene som du skal skrive senere i oppgave 5 skal du kalle de fire statiske metodene i *Sykehus* for å behandle pasientdata. Programmet må kalle forskjellige metoder for kvinner og menn og nå kan du anta at du har med objekter av klassene *KvinneligPasient* og *ManneligPasient* å gjøre.
Det kan derfor være lurt å forstå (og kanskje til å med løse) resten av oppgave 5 før du besvarer oppgave 5A.

Question 5.c
Attached

Exercise 5. (25 points in total + 5 bonus points)

Female and male patients, and processing of patient data using threads.

The first part of this exercise revolves around turning the class *Patient* into an abstract class, subsequently defining two subclasses, *FemalePatient* and *MalePatient*. Then you will process patient data using multiple threads. You will write two thread classes, one class to filter out (*class Sieve*) patients that are in the risk group for contracting a disease, and a class (*class Analysis*) that further analyses these patients.

There shall be one sieve (filter) thread for each priority, that is *MAXPATPRIO + 1* sieve threads. Sieve thread no. *i* must read data from an object of the *PatientPrio* class by calling the *retrieve(i)*method. You must use the *PatientPrio* class even if you have not solved exercise 2.
For every patient retrieved, a static method in the *Hospital* class should be called. These methods return *true* if this patient is in the risk group and thus requires further attention, or *false* in the inverse case. We will assume that these are preexisting methods, written a long time ago and that you may not modify. For males, the method is called *public static boolean inRiskGroupMale(Patient p)*, for women the method is *public static boolean inRiskGroupFemale(Patient p)*. All patients found to be in the risk group are placed in a buffer of the class *AnalysisBuffer*. The sieve threads terminate when the *retrieve()* methods return *null*.

The analysis threads must retrieve patients from the analysis buffer. For each retrieved patient, a static method in the *Hospital* class should be called. In the male case, this method is *public static void potentiallyInfectedMale(Patient p)*, in the female case it is called *public static void potentiallyInfectedFemale(Patient p)*. As above, these methods were written a long time ago and can not be modified, only called from your program. Further processing of these risk patients takes place within these methods. Consequently, the analysis threads do not need to take any further action after these methods return and may retrieve the next patient at that time.

Introduction to exercise 5A (Exercise 5A does not involve any threads).

Copy the code for the *Patient* class from the main introduction section and modify it so that *Patient* becomes an abstract class. Further, write two subclasses, *FemalePatient* and *MalePatient*.
In later parts of exercise 5 you will need to call the four static methods in *Hospital* to process patient data. Your program needs to call different methods for men and women, and you may now assume that you will be dealing with objects of the classes *FemalePatient* and *MalePatient*.
It may be wise to understand (and perhaps even solve) the rest of exercise 5 before answering exercise 5A.

Question 5.c
Attached

Oppgave 5 (totalt 25 poeng + 5 bonuspoeng)

Kvinnelige og Mannlige pasienter og behandling av pasientdata ved hjelp av tråder.

Starten av denne oppgaven dreier seg om å lage klassen *Pasient* abstrakt, og deretter lage to subklasser av denne klassen, kalt *KvinneligPasient* og *MannligPasient*. Deretter skal du behandle pasientdata ved hjelp av tråder. Du skal lage to trådklasser, en klasse for å sile ut (*class Sil*) pasienter som er i faresonen for å få en sykdom, og en klasse (*class Analyse*) som analyserer disse pasientene.

Det skal være en sil-tråd for hver prioritet, altså *MAXPASPRIO + 1* sil-tråder. Sil-tråd nr *i* skal lese data fra et objekt av klassen *PasientPrio* ved å kalle metoden *hentUt(i)*. Du skal bruke klassen *PasientPrio* selv om du ikke har svart på oppgave 2.
For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus* som returnerer *true* hvis dette er en pasient som er i faresonen og derfor skal analyseres, *false* hvis ikke. Dette er metoder som allerede finnes, som ble laget for lenge siden og som du ikke kan forandre. For menn heter metoden *public static boolean iFaresonenMann(Pasient p)*, for kvinner heter metoden *public static boolean iFaresonenKvinne(Pasient p)*.
Alle pasienter som er i faresonen skal legges inn i et buffer av klassen *AnalyseBuffer*.
Sil-trådene terminerer når *hentUt*-metodene returnerer *null*.

Analyse-trådene skal hente pasienter fra analysebufferet. For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus*. For menn skal metoden *public static void kanHaSykdomMann(Pasient p)* kalles, for kvinner skal metoden *public static void kanHaSykdomKvinne(Pasient p)* kalles. Dette er også metoder som ble laget for lenge siden, og som du ikke kan gjøre noe med, bare kalle fra ditt program. Videre behandling av de pasientene som kan ha sykdommen skjer i disse metodene. Det betyr at når disse metodene returnerer skal analyse-trådene ikke gjøre noe mer med denne pasienten, men bare hente ut en ny pasient fra analysebufferet.

Innledning til oppgavene 5A (Oppgave 5A har ikke noe med tråder å gjøre).

Du skal kopiere klassen *Pasient* fra oppgavesettets innledning og modifisere den slik at den blir abstrakt. Deretter skal du lage to subklasser *KvinneligPasient* og *MannligPasient*. I de to trådklassene som du skal skrive senere i oppgave 5 skal du kalle de fire statiske metodene i *Sykehus* for å behandle pasientdata. Programmet må kalle forskjellige metoder for kvinner og menn og nå kan du anta at du har med objekter av klassene *KvinneligPasient* og *ManneligPasient* å gjøre.
Det kan derfor være lurt å forstå (og kanskje til å med løse) resten av oppgave 5 før du besvarer oppgave 5A.

Question 5.d
Attached

Exercise 5. (25 points in total + 5 bonus points)

Female and male patients, and processing of patient data using threads.

The first part of this exercise revolves around turning the class *Patient* into an abstract class, subsequently defining two subclasses, *FemalePatient* and *MalePatient*. Then you will process patient data using multiple threads. You will write two thread classes, one class to filter out (*class Sieve*) patients that are in the risk group for contracting a disease, and a class (*class Analysis*) that further analyses these patients.

There shall be one sieve (filter) thread for each priority, that is *MAXPATPRIO + 1* sieve threads. Sieve thread no. *i* must read data from an object of the *PatientPrio* class by calling the *retrieve(i)*method. You must use the *PatientPrio* class even if you have not solved exercise 2.
For every patient retrieved, a static method in the *Hospital* class should be called. These methods return *true* if this patient is in the risk group and thus requires further attention, or *false* in the inverse case. We will assume that these are preexisting methods, written a long time ago and that you may not modify. For males, the method is called *public static boolean inRiskGroupMale(Patient p)*, for women the method is *public static boolean inRiskGroupFemale(Patient p)*. All patients found to be in the risk group are placed in a buffer of the class *AnalysisBuffer*. The sieve threads terminate when the *retrieve()* methods return *null*.

The analysis threads must retrieve patients from the analysis buffer. For each retrieved patient, a static method in the *Hospital* class should be called. In the male case, this method is *public static void potentiallyInfectedMale(Patient p)*, in the female case it is called *public static void potentiallyInfectedFemale(Patient p)*. As above, these methods were written a long time ago and can not be modified, only called from your program. Further processing of these risk patients takes place within these methods. Consequently, the analysis threads do not need to take any further action after these methods return and may retrieve the next patient at that time.

Introduction to exercise 5A (Exercise 5A does not involve any threads).

Copy the code for the *Patient* class from the main introduction section and modify it so that *Patient* becomes an abstract class. Further, write two subclasses, *FemalePatient* and *MalePatient*.
In later parts of exercise 5 you will need to call the four static methods in *Hospital* to process patient data. Your program needs to call different methods for men and women, and you may now assume that you will be dealing with objects of the classes *FemalePatient* and *MalePatient*.
It may be wise to understand (and perhaps even solve) the rest of exercise 5 before answering exercise 5A.

Question 5.d
Attached

Oppgave 5 (totalt 25 poeng + 5 bonuspoeng)

Kvinnelige og Mannlige pasienter og behandling av pasientdata ved hjelp av tråder.

Starten av denne oppgaven dreier seg om å lage klassen *Pasient* abstrakt, og deretter lage to subklasser av denne klassen, kalt *KvinneligPasient* og *MannligPasient*. Deretter skal du behandle pasientdata ved hjelp av tråder. Du skal lage to trådklasser, en klasse for å sile ut (*class Sil*) pasienter som er i faresonen for å få en sykdom, og en klasse (*class Analyse*) som analyserer disse pasientene.

Det skal være en sil-tråd for hver prioritet, altså *MAXPASPRIO + 1* sil-tråder. Sil-tråd nr *i* skal lese data fra et objekt av klassen *PasientPrio* ved å kalle metoden *hentUt(i)*. Du skal bruke klassen *PasientPrio* selv om du ikke har svart på oppgave 2.
For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus* som returnerer *true* hvis dette er en pasient som er i faresonen og derfor skal analyseres, *false* hvis ikke. Dette er metoder som allerede finnes, som ble laget for lenge siden og som du ikke kan forandre. For menn heter metoden *public static boolean iFaresonenMann(Pasient p)*, for kvinner heter metoden *public static boolean iFaresonenKvinne(Pasient p)*.
Alle pasienter som er i faresonen skal legges inn i et buffer av klassen *AnalyseBuffer*.
Sil-trådene terminerer når *hentUt*-metodene returnerer *null*.

Analyse-trådene skal hente pasienter fra analysebufferet. For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus*. For menn skal metoden *public static void kanHaSykdomMann(Pasient p)* kalles, for kvinner skal metoden *public static void kanHaSykdomKvinne(Pasient p)* kalles. Dette er også metoder som ble laget for lenge siden, og som du ikke kan gjøre noe med, bare kalle fra ditt program. Videre behandling av de pasientene som kan ha sykdommen skjer i disse metodene. Det betyr at når disse metodene returnerer skal analyse-trådene ikke gjøre noe mer med denne pasienten, men bare hente ut en ny pasient fra analysebufferet.

Innledning til oppgavene 5A (Oppgave 5A har ikke noe med tråder å gjøre).

Du skal kopiere klassen *Pasient* fra oppgavesettets innledning og modifisere den slik at den blir abstrakt. Deretter skal du lage to subklasser *KvinneligPasient* og *MannligPasient*. I de to trådklassene som du skal skrive senere i oppgave 5 skal du kalle de fire statiske metodene i *Sykehus* for å behandle pasientdata. Programmet må kalle forskjellige metoder for kvinner og menn og nå kan du anta at du har med objekter av klassene *KvinneligPasient* og *ManneligPasient* å gjøre.
Det kan derfor være lurt å forstå (og kanskje til å med løse) resten av oppgave 5 før du besvarer oppgave 5A.

Question 5.e
Attached

Exercise 5. (25 points in total + 5 bonus points)

Female and male patients, and processing of patient data using threads.

The first part of this exercise revolves around turning the class *Patient* into an abstract class, subsequently defining two subclasses, *FemalePatient* and *MalePatient*. Then you will process patient data using multiple threads. You will write two thread classes, one class to filter out (*class Sieve*) patients that are in the risk group for contracting a disease, and a class (*class Analysis*) that further analyses these patients.

There shall be one sieve (filter) thread for each priority, that is *MAXPATPRIO + 1* sieve threads. Sieve thread no. *i* must read data from an object of the *PatientPrio* class by calling the *retrieve(i)*method. You must use the *PatientPrio* class even if you have not solved exercise 2.
For every patient retrieved, a static method in the *Hospital* class should be called. These methods return *true* if this patient is in the risk group and thus requires further attention, or *false* in the inverse case. We will assume that these are preexisting methods, written a long time ago and that you may not modify. For males, the method is called *public static boolean inRiskGroupMale(Patient p)*, for women the method is *public static boolean inRiskGroupFemale(Patient p)*. All patients found to be in the risk group are placed in a buffer of the class *AnalysisBuffer*. The sieve threads terminate when the *retrieve()* methods return *null*.

The analysis threads must retrieve patients from the analysis buffer. For each retrieved patient, a static method in the *Hospital* class should be called. In the male case, this method is *public static void potentiallyInfectedMale(Patient p)*, in the female case it is called *public static void potentiallyInfectedFemale(Patient p)*. As above, these methods were written a long time ago and can not be modified, only called from your program. Further processing of these risk patients takes place within these methods. Consequently, the analysis threads do not need to take any further action after these methods return and may retrieve the next patient at that time.

Introduction to exercise 5A (Exercise 5A does not involve any threads).

Copy the code for the *Patient* class from the main introduction section and modify it so that *Patient* becomes an abstract class. Further, write two subclasses, *FemalePatient* and *MalePatient*.
In later parts of exercise 5 you will need to call the four static methods in *Hospital* to process patient data. Your program needs to call different methods for men and women, and you may now assume that you will be dealing with objects of the classes *FemalePatient* and *MalePatient*.
It may be wise to understand (and perhaps even solve) the rest of exercise 5 before answering exercise 5A.

Question 5.e
Attached

Oppgave 5 (totalt 25 poeng + 5 bonuspoeng)

Kvinnelige og Mannlige pasienter og behandling av pasientdata ved hjelp av tråder.

Starten av denne oppgaven dreier seg om å lage klassen *Pasient* abstrakt, og deretter lage to subklasser av denne klassen, kalt *KvinneligPasient* og *MannligPasient*. Deretter skal du behandle pasientdata ved hjelp av tråder. Du skal lage to trådklasser, en klasse for å sile ut (*class Sil*) pasienter som er i faresonen for å få en sykdom, og en klasse (*class Analyse*) som analyserer disse pasientene.

Det skal være en sil-tråd for hver prioritet, altså *MAXPASPRIO + 1* sil-tråder. Sil-tråd nr *i* skal lese data fra et objekt av klassen *PasientPrio* ved å kalle metoden *hentUt(i)*. Du skal bruke klassen *PasientPrio* selv om du ikke har svart på oppgave 2.
For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus* som returnerer *true* hvis dette er en pasient som er i faresonen og derfor skal analyseres, *false* hvis ikke. Dette er metoder som allerede finnes, som ble laget for lenge siden og som du ikke kan forandre. For menn heter metoden *public static boolean iFaresonenMann(Pasient p)*, for kvinner heter metoden *public static boolean iFaresonenKvinne(Pasient p)*.
Alle pasienter som er i faresonen skal legges inn i et buffer av klassen *AnalyseBuffer*.
Sil-trådene terminerer når *hentUt*-metodene returnerer *null*.

Analyse-trådene skal hente pasienter fra analysebufferet. For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus*. For menn skal metoden *public static void kanHaSykdomMann(Pasient p)* kalles, for kvinner skal metoden *public static void kanHaSykdomKvinne(Pasient p)* kalles. Dette er også metoder som ble laget for lenge siden, og som du ikke kan gjøre noe med, bare kalle fra ditt program. Videre behandling av de pasientene som kan ha sykdommen skjer i disse metodene. Det betyr at når disse metodene returnerer skal analyse-trådene ikke gjøre noe mer med denne pasienten, men bare hente ut en ny pasient fra analysebufferet.

Innledning til oppgavene 5A (Oppgave 5A har ikke noe med tråder å gjøre).

Du skal kopiere klassen *Pasient* fra oppgavesettets innledning og modifisere den slik at den blir abstrakt. Deretter skal du lage to subklasser *KvinneligPasient* og *MannligPasient*. I de to trådklassene som du skal skrive senere i oppgave 5 skal du kalle de fire statiske metodene i *Sykehus* for å behandle pasientdata. Programmet må kalle forskjellige metoder for kvinner og menn og nå kan du anta at du har med objekter av klassene *KvinneligPasient* og *ManneligPasient* å gjøre.
Det kan derfor være lurt å forstå (og kanskje til å med løse) resten av oppgave 5 før du besvarer oppgave 5A.

Question 5.f
Attached

Exercise 5. (25 points in total + 5 bonus points)

Female and male patients, and processing of patient data using threads.

The first part of this exercise revolves around turning the class *Patient* into an abstract class, subsequently defining two subclasses, *FemalePatient* and *MalePatient*. Then you will process patient data using multiple threads. You will write two thread classes, one class to filter out (*class Sieve*) patients that are in the risk group for contracting a disease, and a class (*class Analysis*) that further analyses these patients.

There shall be one sieve (filter) thread for each priority, that is *MAXPATPRIO + 1* sieve threads. Sieve thread no. *i* must read data from an object of the *PatientPrio* class by calling the *retrieve(i)*method. You must use the *PatientPrio* class even if you have not solved exercise 2.
For every patient retrieved, a static method in the *Hospital* class should be called. These methods return *true* if this patient is in the risk group and thus requires further attention, or *false* in the inverse case. We will assume that these are preexisting methods, written a long time ago and that you may not modify. For males, the method is called *public static boolean inRiskGroupMale(Patient p)*, for women the method is *public static boolean inRiskGroupFemale(Patient p)*. All patients found to be in the risk group are placed in a buffer of the class *AnalysisBuffer*. The sieve threads terminate when the *retrieve()* methods return *null*.

The analysis threads must retrieve patients from the analysis buffer. For each retrieved patient, a static method in the *Hospital* class should be called. In the male case, this method is *public static void potentiallyInfectedMale(Patient p)*, in the female case it is called *public static void potentiallyInfectedFemale(Patient p)*. As above, these methods were written a long time ago and can not be modified, only called from your program. Further processing of these risk patients takes place within these methods. Consequently, the analysis threads do not need to take any further action after these methods return and may retrieve the next patient at that time.

Introduction to exercise 5A (Exercise 5A does not involve any threads).

Copy the code for the *Patient* class from the main introduction section and modify it so that *Patient* becomes an abstract class. Further, write two subclasses, *FemalePatient* and *MalePatient*.
In later parts of exercise 5 you will need to call the four static methods in *Hospital* to process patient data. Your program needs to call different methods for men and women, and you may now assume that you will be dealing with objects of the classes *FemalePatient* and *MalePatient*.
It may be wise to understand (and perhaps even solve) the rest of exercise 5 before answering exercise 5A.

Question 5.f
Attached

Oppgave 5 (totalt 25 poeng + 5 bonuspoeng)

Kvinnelige og Mannlige pasienter og behandling av pasientdata ved hjelp av tråder.

Starten av denne oppgaven dreier seg om å lage klassen *Pasient* abstrakt, og deretter lage to subklasser av denne klassen, kalt *KvinneligPasient* og *MannligPasient*. Deretter skal du behandle pasientdata ved hjelp av tråder. Du skal lage to trådklasser, en klasse for å sile ut (*class Sil*) pasienter som er i faresonen for å få en sykdom, og en klasse (*class Analyse*) som analyserer disse pasientene.

Det skal være en sil-tråd for hver prioritet, altså *MAXPASPRIO + 1* sil-tråder. Sil-tråd nr *i* skal lese data fra et objekt av klassen *PasientPrio* ved å kalle metoden *hentUt(i)*. Du skal bruke klassen *PasientPrio* selv om du ikke har svart på oppgave 2.
For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus* som returnerer *true* hvis dette er en pasient som er i faresonen og derfor skal analyseres, *false* hvis ikke. Dette er metoder som allerede finnes, som ble laget for lenge siden og som du ikke kan forandre. For menn heter metoden *public static boolean iFaresonenMann(Pasient p)*, for kvinner heter metoden *public static boolean iFaresonenKvinne(Pasient p)*.
Alle pasienter som er i faresonen skal legges inn i et buffer av klassen *AnalyseBuffer*.
Sil-trådene terminerer når *hentUt*-metodene returnerer *null*.

Analyse-trådene skal hente pasienter fra analysebufferet. For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus*. For menn skal metoden *public static void kanHaSykdomMann(Pasient p)* kalles, for kvinner skal metoden *public static void kanHaSykdomKvinne(Pasient p)* kalles. Dette er også metoder som ble laget for lenge siden, og som du ikke kan gjøre noe med, bare kalle fra ditt program. Videre behandling av de pasientene som kan ha sykdommen skjer i disse metodene. Det betyr at når disse metodene returnerer skal analyse-trådene ikke gjøre noe mer med denne pasienten, men bare hente ut en ny pasient fra analysebufferet.

Innledning til oppgavene 5A (Oppgave 5A har ikke noe med tråder å gjøre).

Du skal kopiere klassen *Pasient* fra oppgavesettets innledning og modifisere den slik at den blir abstrakt. Deretter skal du lage to subklasser *KvinneligPasient* og *MannligPasient*. I de to trådklassene som du skal skrive senere i oppgave 5 skal du kalle de fire statiske metodene i *Sykehus* for å behandle pasientdata. Programmet må kalle forskjellige metoder for kvinner og menn og nå kan du anta at du har med objekter av klassene *KvinneligPasient* og *ManneligPasient* å gjøre.
Det kan derfor være lurt å forstå (og kanskje til å med løse) resten av oppgave 5 før du besvarer oppgave 5A.

Question 5.g
Attached

Exercise 5. (25 points in total + 5 bonus points)

Female and male patients, and processing of patient data using threads.

The first part of this exercise revolves around turning the class *Patient* into an abstract class, subsequently defining two subclasses, *FemalePatient* and *MalePatient*. Then you will process patient data using multiple threads. You will write two thread classes, one class to filter out (*class Sieve*) patients that are in the risk group for contracting a disease, and a class (*class Analysis*) that further analyses these patients.

There shall be one sieve (filter) thread for each priority, that is *MAXPATPRIO + 1* sieve threads. Sieve thread no. *i* must read data from an object of the *PatientPrio* class by calling the *retrieve(i)*method. You must use the *PatientPrio* class even if you have not solved exercise 2.
For every patient retrieved, a static method in the *Hospital* class should be called. These methods return *true* if this patient is in the risk group and thus requires further attention, or *false* in the inverse case. We will assume that these are preexisting methods, written a long time ago and that you may not modify. For males, the method is called *public static boolean inRiskGroupMale(Patient p)*, for women the method is *public static boolean inRiskGroupFemale(Patient p)*. All patients found to be in the risk group are placed in a buffer of the class *AnalysisBuffer*. The sieve threads terminate when the *retrieve()* methods return *null*.

The analysis threads must retrieve patients from the analysis buffer. For each retrieved patient, a static method in the *Hospital* class should be called. In the male case, this method is *public static void potentiallyInfectedMale(Patient p)*, in the female case it is called *public static void potentiallyInfectedFemale(Patient p)*. As above, these methods were written a long time ago and can not be modified, only called from your program. Further processing of these risk patients takes place within these methods. Consequently, the analysis threads do not need to take any further action after these methods return and may retrieve the next patient at that time.

Introduction to exercise 5A (Exercise 5A does not involve any threads).

Copy the code for the *Patient* class from the main introduction section and modify it so that *Patient* becomes an abstract class. Further, write two subclasses, *FemalePatient* and *MalePatient*.
In later parts of exercise 5 you will need to call the four static methods in *Hospital* to process patient data. Your program needs to call different methods for men and women, and you may now assume that you will be dealing with objects of the classes *FemalePatient* and *MalePatient*.
It may be wise to understand (and perhaps even solve) the rest of exercise 5 before answering exercise 5A.

Question 5.g
Attached

Oppgave 5 (totalt 25 poeng + 5 bonuspoeng)

Kvinnelige og Mannlige pasienter og behandling av pasientdata ved hjelp av tråder.

Starten av denne oppgaven dreier seg om å lage klassen *Pasient* abstrakt, og deretter lage to subklasser av denne klassen, kalt *KvinneligPasient* og *MannligPasient*. Deretter skal du behandle pasientdata ved hjelp av tråder. Du skal lage to trådklasser, en klasse for å sile ut (*class Sil*) pasienter som er i faresonen for å få en sykdom, og en klasse (*class Analyse*) som analyserer disse pasientene.

Det skal være en sil-tråd for hver prioritet, altså *MAXPASPRIO + 1* sil-tråder. Sil-tråd nr *i* skal lese data fra et objekt av klassen *PasientPrio* ved å kalle metoden *hentUt(i)*. Du skal bruke klassen *PasientPrio* selv om du ikke har svart på oppgave 2.
For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus* som returnerer *true* hvis dette er en pasient som er i faresonen og derfor skal analyseres, *false* hvis ikke. Dette er metoder som allerede finnes, som ble laget for lenge siden og som du ikke kan forandre. For menn heter metoden *public static boolean iFaresonenMann(Pasient p)*, for kvinner heter metoden *public static boolean iFaresonenKvinne(Pasient p)*.
Alle pasienter som er i faresonen skal legges inn i et buffer av klassen *AnalyseBuffer*.
Sil-trådene terminerer når *hentUt*-metodene returnerer *null*.

Analyse-trådene skal hente pasienter fra analysebufferet. For hver pasient som hentes ut skal det kalles en statisk metode i klassen *Sykehus*. For menn skal metoden *public static void kanHaSykdomMann(Pasient p)* kalles, for kvinner skal metoden *public static void kanHaSykdomKvinne(Pasient p)* kalles. Dette er også metoder som ble laget for lenge siden, og som du ikke kan gjøre noe med, bare kalle fra ditt program. Videre behandling av de pasientene som kan ha sykdommen skjer i disse metodene. Det betyr at når disse metodene returnerer skal analyse-trådene ikke gjøre noe mer med denne pasienten, men bare hente ut en ny pasient fra analysebufferet.

Innledning til oppgavene 5A (Oppgave 5A har ikke noe med tråder å gjøre).

Du skal kopiere klassen *Pasient* fra oppgavesettets innledning og modifisere den slik at den blir abstrakt. Deretter skal du lage to subklasser *KvinneligPasient* og *MannligPasient*. I de to trådklassene som du skal skrive senere i oppgave 5 skal du kalle de fire statiske metodene i *Sykehus* for å behandle pasientdata. Programmet må kalle forskjellige metoder for kvinner og menn og nå kan du anta at du har med objekter av klassene *KvinneligPasient* og *ManneligPasient* å gjøre.
Det kan derfor være lurt å forstå (og kanskje til å med løse) resten av oppgave 5 før du besvarer oppgave 5A.