KANDIDAT

10085

PRØVE

TDAT1001 A Programmering grunnkurs

Emnekode	TDAT1001
Vurderingsform	Skriftlig eksamen
Starttid	20.12.2018 08:00
Sluttid	20.12.2018 12:00
Sensurfrist	21.01.2019 23:59
PDF opprettet	22.11.2019 10:45

Informasjon

Oppgave	Oppgavetype
i	Dokument

Oppgave 1

Oppgave	Oppgavetype
2.1	Langsvar
2.2	Programmering
2.3	Programmering
2.4	Programmering

Vedlegg

Oppgave	Oppgavetype
i	Dokument
i	Dokument

2

Case-beskrivelse

Du skal lage en applikasjon for Soppkontrollen. Soppkontrollen har oversikt over alle sopparter som finnes i norsk fauna og er behjelpelig med å klassifisere sopp som turgåere og sankere har observert og/eller plukket og tatt med seg hjem.

Du skal i denne eksamensoppgaven jobbe med 3 ulike klasser:

- Soppart
- Soppregister
- Klientprogram

Soppart
En soppart har et navn, beskrivelse og informasjon om soppen er giftig eller ikke.
Klassen skal være immutabel.
Eksempel på hva som kan registreres om en soppart:
Navn: `Rød fluesopp`
Beskrivelse: `Rød sopp med prikker. Du finner den i skog med bjørk og gran.`
Giftig: `Ja`

Soppregister
Har en liste over alle sopparter. Når det opprettes et nytt soppregister skal det opprettes et tomt register med plass til 10 ulike sopparter. Dersom det registreres flere enn 10 sopparter (listen er full) skal den utvides med 10 nye plasser.

Eksempel på en liste med fire registrerte sopparter:

```
Registrerte Sopparter (Navn Beskrivelse Spiselig):
Rød fluesopp Rød sopp med prikker. Du finner den i skog med bjørk og gran. Giftig
Grønn fluesopp Grønn sopp. Næringsrik løvskog med eik, men også med bøk og hassel.  Giftig
Kantarell Hele soppen er eggeplommegul, kjøttfull og traktformet med gaffelgrenete nedløpende
ribber. Trives best i moserik blåbærgranskog og i gammel bjørkeskog. Matsopp
Kongesjampinjong Hatten er først nesten kulerund med avflatet topp, senere mer hvelvet.
Næringsrik jord i hager og parker. Matsopp
```

Klientprogram
Et menystyrt program for å teste klasser og metoder og har også klassemetoder for å lese/ skrive hele soppregisteret til fil.

For å forenkle oppgaven noe så er soppartene enten spiselige eller giftige (matsopp/ giftig).

1 Oppgave 1 - Modellering

a. Les casebeskrivelse og hele oppgavesettet og lag ett utvidet klassediagram for alle klassene.
b. Vis i klassediagrammet hvordan du mener de ulike klassene bør samarbeide med

hverandre. Begrunn svaret ditt i tekstfeltet nedenfor.

Med unntak av begrunnelsen i oppgave 1b, løses oppgave 1 på eget ark.

Skriv ditt svar her...

Jeg har valgt å bruke komposisjon mellom klassen Soppart og Soppregister. Grunnen til det er fordi vi ikke ønsker at noen skal kunne endre registeret vårt. Og siden soppart er immutabel gjør det lettere å holde orden på registeret. På denne måten deler vi kun ut kopier av registert og ikke selve registeret som gjør at det er mindre sannsynlig at registeret blir korrupt. Dette fører selvfølgelig til mer prossesseringskraft, siden listen oppretter en egen kopi av hver soppart, men det gjør at det kun er en referanse til hvert objekt.

Maks poeng: 10

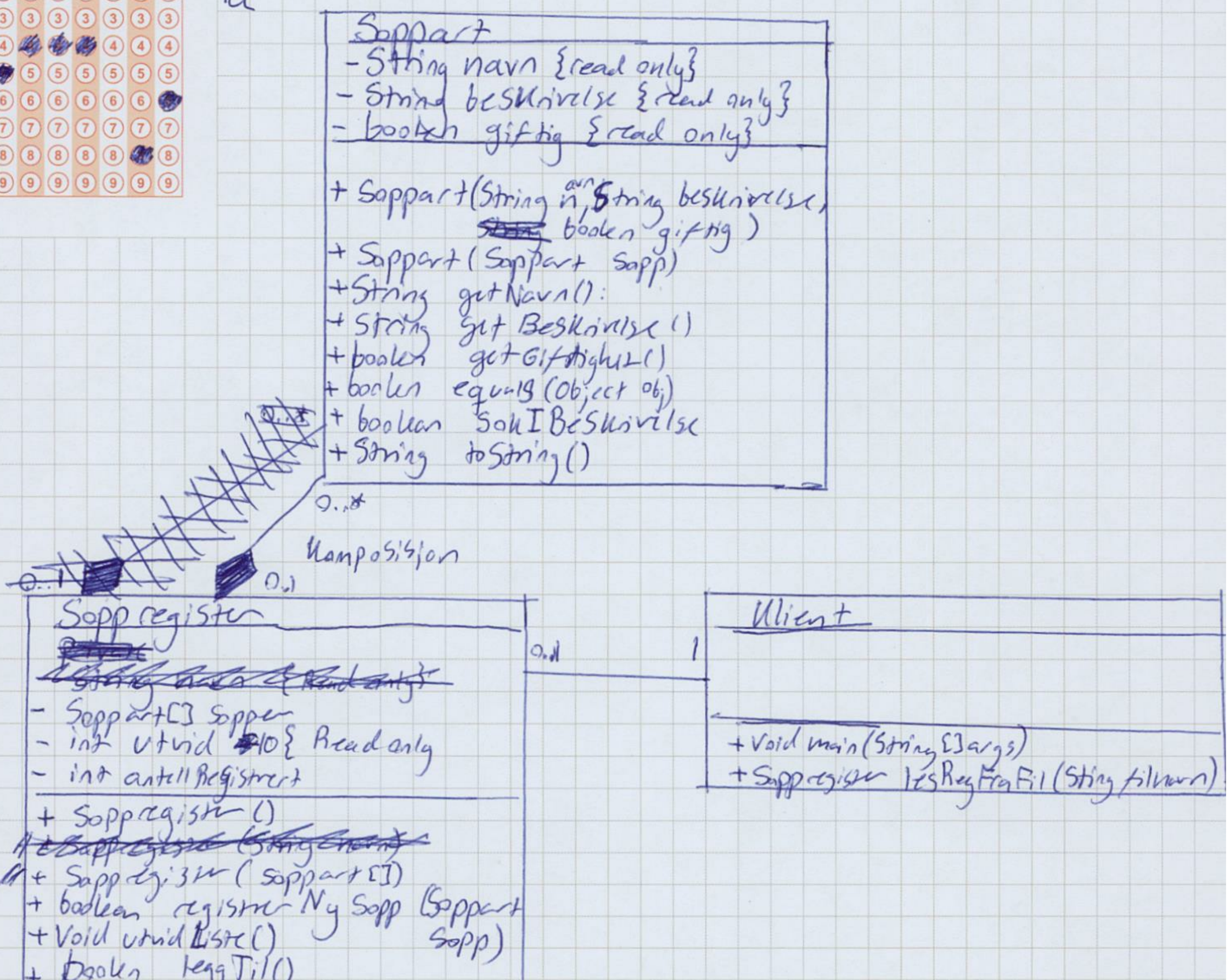
Knytte håndtegninger til denne oppgaven?

Bruk følgende kode:

5 4 4 4 0 8 6

Question Code Oppgavekode	Date Dato	Subject code Emnekode	Candidate ID KandidatID	Question nr Oppgavenr	Page number Sidetall
5 4 4 4 0 8 6	20.12.18	TDAT1001	10085	oppg 1 a/b	1/2

1a



2 Oppgave 2 - Klassen Soppart

sett opp klassens objektvariabler

a. lag to ulike konstruktører, en som tar inn verdier til alle klassens objektvariabler og en konstruktør som tar inn et objekt av typen Soppart.
b. sett opp klassens tilgangsmetoder.
c. Lag en metode som sjekker om en soppart er lik en annet. To sopparter er like dersom navnet er likt.
d. Lag en metode som sjekker om en gitt søkestreng finnes i beskrivelsen av sopparten. Metoden skal returnere true eller false avhengig av om søkestrengen finnes som en del av soppartens beskrivelse.

Skriv ditt svar her...

```java
//siden det skal leses til fil, må classen være serializabe
class Soppart implements java.io.Serializable{ //ikke nødvendig, mer for min egen del

    //Klassens immutable objektvariabler
    private final String navn;
    private final String beskrivelse;
    private final boolean giftig;

    //klassens ene konstrultør, med objektvariabel som parameter
    public Soppart(String navn, String beskrivelse, boolean giftig){
        this.navn = navn;
        this.beskrivelse = beskrivelse;
        this.giftig = giftig;
    }

    //Klassens andre konstruktør, med Soppart objekt som parameter.
    //Bruker parameter sine tilgangsmetoder til å tildele this sine
    //variabler en verdi lik parameter-objektet
    public Soppart(Soppart sopp){
        this.navn = sopp.getNavn();
        this.beskrivelse = sopp.getBeskrivelse();
        this.gifitg = sopp.getGiftighet();
    }

    public String getNavn(){
        return this.navn;
    }

    public String getBeskrivelse(){
        return this.beskrivelse;
    }

    public boolean getGiftighet(){
        return this.giftig;
    }

    //equals metode som sjekker om to objekter er like, basert på deres navn
    public boolean equals(Object obj){
        //hvis objektet er lik dette objektet er de like
        if(obj == this) return true;
        //hvis objektet ikke er av klassetypen sopp er de ikke like
        if(!(obj instanceof Soppart)) return false;
        //caster objektet til Soppart og bruker .equals()-metoden for å sammenligne navn stingen
        Soppart sopp = (Soppart) obj;
        return this.navn.equals(sopp.getNavn());
    }

    //Første tanke: Sjekker om sokestringen er lik en av setningene i beskrivelsen ved å splitte
        beskrivelsen for hvert "." og sammenligner med .equals() metode
    /*
    public boolean sokIBeskrivelse(String sokeString){
        String[] setning = this.beskrivelse.split(".");
        for(String s : setning){
            if(sokeString.equals(s)) return true;
        }
        return false;
    }
    */

    //Etter spm til faglærer ble oppgaven oppklart ved at søkestrengen er hele beskrivelsen, så
        sammenligner med om den er lik beskrivelsen. Antar at store/små bokstaver er like for
        sokestreng og beskrivelse.
    public boolean sokIBeskrivelse(String sokeString){
        if(sokeString.equals(this.beskrivelse)) return true;
        return false;
    }
    //Tostring for finere kode i soppregister klassen, ikke en del av oppgaven
    public String toString(){
        String output = this.navn + " " + this.beskrivelse + " ";
        if(this.giftig){
            outut += "Giftig\n"
        }else{
            output += "Matsopp\n"
        }
        return output;
    }

}
```

Maks poeng: 10

Knytte håndtegninger til denne oppgaven?

3 Oppgave 3 - Klassen Soppregister

a. Sett opp klassens objektvariabler
b. Sett opp klassens konstruktør
c. Lag en metode som registrer en ny soppart. Ny soppart kan kun registreres dersom den ikke er registrert fra før. Dersom det ikke er plass i listen skal denne utvides med 10 nye plasser.
d. Lag en metode som returnerer alle matsoppene.
e. Lag en metode som returnerer en tekststreng med all informasjon om alle registrerte sopparter. Eks på utskrift: se nedenfor.
f. Lag en metode som søker gjennom alle registrerte sopparter om en gitt søkestreng finnes i beskrivelsen av den enkelte soppart. Metoden skal returnere en teststreng med informasjon om alle sopparter som svarer til søket

```
Alle registrerte Sopparter (Navn Beskrivelse Spiselig):
1: Rød fluesopp Rød sopp med prikker. Du finner den i skog med bjørk og gran. Giftig
2: Grønn fluesopp Grønn sopp. Næringsrik løvskog med eik, men også med bøk og hassel. Giftig
3: Kantarell Hele soppen er eggeplommegul, kjøttfull og traktformet med gaffelgrenete nedløpende ribber. Trives best i moserik blåbærgranskog og i gammel bjørkeskog. Matsopp
4: Kongesjampinjong Hatten er først nesten kulerund med avflatet topp. Næringsrik jord i hager og parker. Matsopp
```

Skriv ditt svar her...

```java
//siden det skal leses til fil, må classen være serializabe
class Soppregister implements java.io.Serializable{
    private Soppart[] sopper;
    //definerer en int = 10, utifra informasjon gitt i oppgavetekst
    private final int utvid = 10;
    //oppretter en int for å holde orden over antall registrerte
    private int antallRegistrert;

    public Soppregister(){
        this.sopper = new Soppart[utvid];
        this.antallRegistrert = 0;
    }

    //Siden det står i oppgaveteksten at det skal både leses og skrives til fil, kan man definere en ny Soppregister objekt med en tidligere liste
    //Velger komposisjon som beskrevet tidligere og derfor tar en dyp kopi av listen for å ha mest mulig kontroll
    /*
    public Soppregister(Soppart[] liste){
        this.sopper = new Soppart[liste.length];
        for(int i = 0; i < liste.length; i++){
            this.sopper[i] = new Soppart(liste[i].getNavn(), liste[i].getBeskrivelse(), liste[i].getGiftighet())
        }
        this.antallRegistrert = liste.length;
    }
    */

    public boolean registrerNySopp(Soppart sopp){
        //sjekker om objektet er null;
        if(sopp == null) return false;
        //Sjekker om listen er lang nok, hvis ikke -> utvid med 10;
        //hvis det er ingenting i listen, bare legg til;
        if(antallRegistrerte == 0) return leggTil(sopp);
        //bruker for:each loop til å gå gjennom listen, sjekker om den er lik en annen sopp i listen
        for(Soppart s : sopper){
            if(s != null){
                if(sopp.equals(s){
                    return false;
                }
            }else{
                break;
            }
        }
        //utvidListe er egenmetode
        if(antallRegistrerte == sopper.length) utvidListe();
        //ettersom den har bestått alle testene legger jeg den inn i listen.
        return leggTil(sopp);
    }
```

```
47
48      public void utvidListe(){
49          Soppart[] kopi = new Soppart[sopper.length + this.utvid];
50          for(int i = 0; i < antallRegistrert; i++){
51              kopi[i] = sopper[i];
52          }
53          sopper = kopi;
54      }
55
56      public boolean leggTil(Soppart sopp){
57          sopper[antallRegistrert] = new Soppart(sopp.getNavn(), sopp.getBeskrivelse(), sopp
                .getGiftighet());
58          antallRegistrert++;
59          return true;
60      }
61
62      public Soppart[] spiseligSopper(){
63          //velger å definere en like stor liste som den opprinnelige listen, fordi det er mer tungt
                for maskinen å utvide listen, enn å ha en liste med tomme plasser.
64          Soppart[] spiselig = new Soppart[antallRegistrert];
65          int spiseligRegistrert = 0;
66
67          for(int i = 0; i < antallRegistrert; i++){
68              //siden jeg definerte giftighet basert på true/false, vil false gi verdi at den er
                    spiselig
69              if(!(sopper[i].getGiftighet)){
70                  spiselig[spiseligRegistrert] = new Soppart(sopper[i].getNavn(), sopper[i]
                        .getBeskrivelse(), sopper[i].getGiftighet());
71                  spiseligRegistrert++;
72              }
73          }
74          return trimListe(spiselig, spiseligRegistrert);
75      }
76      //trimmer bort de tomme plassene i listen, slik at metoden over spiseligSopp returnerer en full
            liste
77      public Soppart[] trimListe(Soppart[] spiseligListe, int reg){
78          Soppart[] kopi = new Soppart[reg-1];
79          for(int i = 0; i < reg; i++){
80              kopi[i] = spiseligListe[i];
81          }
82          return kopi;
83      }
84
85      public String toString(){
86          String output = "Registrerte Sopparter (Navn Beskrivelse Spiselig): \n";
87          for(int i = 0; i < antallRegistret; i++){
88              output += (i+1) + ": ";
89              //Opprettet en toString i classen Soppart (Side 2, og nederst av siden)
90              output += sopper[i].toString();
91          }
92          return output;
93      }
94
95      public String sokeIBeskrivelse(String sokeString){
96          String output = "Sopper med matchende beskrivelse:\n";
97          int teller = 1;
98          for(int i = 0; i < antallRegistret; i++){
99              if(sopper[i].sokIBeskrivelse(sokeString){
100                 output += teller + ": ";
101                 output += sopper[i].toString();
102                 teller++;
103             }
104         }
105         return output;
106     }
107 }
108 //samme som ligger på side 2, men siden den blir brukt i denne oppgaven valgte jeg å lime den inn
        her også
109 class Soppart{
110     public String toString(){
111         String output = this.navn + " " + this.beskrivelse + " ";
112         if(this.giftig){
113             outut += "Giftig\n"
114         }else{
115             output += "Matsopp\n"
116         }
117         return output;
118     }
119 }
120
```

Maks poeng: 10

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

9 7 5 0 5 6 8

4 Oppgave 4 - Klientprogram

a. lag en klassemetode som leser et objekt av typen Soppregister fra ei fil og returner dette.
b. I vedlegg 2 finner du rammeverket til et menystyrt program som gir bruker følgende valgmuligheter: List alle registrerte sopparter, List alle matsopper, Søk, Legg til ny Soppart og Avslutt. Idenne oppgaven skal du legge inn kode for to (2) av menyvalgene:
    1. Legg inn ny soppart
    2. Søk
c. Lag et UML aktivitets diagram for menyvalget «List alle matsopper».

Oppgave 4c) løses på utdelt ark.

Skriv ditt svar her...

```
class klient{
    public Soppregister lesRegFraFil(String filNavn){
        java.io.FileInputStream fis = null;
        java.io.ObjectInputStream ois = null;
        try{
            fis = new FileInputStream(filNavn);
            ois = new ObjectInputStream(fis);
            Soppregister register = (Soppregister) ois.readObject();
            return register;
        }catch(FileNotFoundException e){
            e.printStackTrace();
            return null;
        }
        catch(IOException e){
            e.printStackTrace();
            return null;
        }
        catch(NullPointerException e){
            e.printStackTrace();
            return null;
        }catch(ClassNotFoundException e){
            e.printStackTrace()
        }finally{
            try{
                fis.close();
                ois.close();
            }catch(IOException e){
                e.printStackTrace();
            }
        }
    }


    //oppgave 2.1
    case 2:
        String navn = showInputDialog("Navn til soppen");
        String beskrivelse = showInputDialog("Legg til beskrivelse");
        boolean giftig;
        int valg = showConfirmDialog(null, "Er soppen giftig", "Eksamen 2018", YES_NO_OPTION);
        if(valg == 0){
            giftig = true;
        }else{
            giftig = false;
        }
        if(register.registrerNySopp(new Soppart(navn, beskrivelse, giftig)){
            showMessageDialog(null, "Registrering var en suksess");
        }else{
            showMessageDialog(null, "Skjedde en feil i registrering");
        }
    break;

    //2.2
    case 3:
        String sokeString = showInputDialog("Vennligst oppgi onsket sokestring");
        String resultat = register.sokIBeskrivelse(sokeString);
        showMessageDialog(null, resultat);
    break;
}
```

Maks poeng: 10

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

1 6 3 7 4 3 5

Question Code Oppgavekode	Date Dato	Subject code Emnekode	Candidate ID KandidatID	Question nr Oppgavenr	Page number Sidetall
1 6 3 7 4 3 5	20.12.16	TDAT 1001	10085	oppg 4 c	2/2