Universitetet i Sørøst-Norge

EKSAMENSFORSIDE

Skoleeksamen med tilsyn

Generell informasjon om eksamen:

Emnekode: OBJ2000

Emnenavn: Objektorientert programmering 1

Dato: 17.12.2024

Start klokkeslett: 09:00

Antall timer: 4 timer

Emneansvarlig: Tomas Attila Paulsen Olaj

Campus:
Ringerike

Fakultetet: HH

Antall oppgaver: Del 1: 20, Del 2: 40, Del 3: 2 (totalt 62 oppgaver)

Totalt antall sider (inkludert forside): 48

Tillatte hjelpemidler (jfr. emneplan):

Alle trykte og skrevne hjelpemidler tillatt.

Merknader:

Oppgavesettet er inndelt i 3 deler med forskjellige typer oppgaver. Det er på hver del angitt hvor mye denne teller av totalen. Karakter fastsettes dog på basis av en helhetsvurdering av besvarelsen. LYKKE TIL!

EN RIKTIG GOD JUL OG ET GODT NYTT ÅR TIL ALLE SAMMEN!

Del 1 – Flervalg om Java-kunnskap (20 %)
Del 2 – Flervalg om Java-kodeforståelse (40 %)
Del 3 – Case (Java programmeringsoppgave/OOP-design (40 %)

Ved eksamen på papir kryss av for type eksamenspapir:
Ruter ☐ Linjer ☐

KANDIDATEN MÅ SELV KONTROLLERE AT OPPGAVESETTET ER FULLSTENDIG

Del 1 – Informasjon

Denne delen består av 20 flervalg-oppgaver som hver gir 1 poeng (1 %) ved riktig svar, og -0,25 poeng i minus ved feil svar (total minus poeng som kan oppnås er -5). Du skal velge kun ***ett*** alternativ. Det kan være at ett alternativ er mer korrekt enn de andre alternativene, og det mest korrekte alternativet vil gi poeng. Hvis kandidaten foreslår mer enn ett riktig svar, så vil oppgaven ikke godkjennes.

Du kan besvare oppgaven i WISEflow ved å [redacted],

[redacted]

skrive opp en liste med: *oppgavenummer+korrekt løsningsforslag: a,b,c eller d*,

f.eks. 1.b, 2.c, 3.a, osv. til og med oppgavenummer 20.

Del 1 – Flervalg om Java-kunnskap (20 %)

OPPGAVER

1. Kan Java kjøre på hvilken som helst maskin?

A. Ja, så lenge maskinen har en nettleser.
B. Nei, Java er begrenset til spesifikke operativsystemer.
C. Ja, så lenge maskinen har en Java Virtual Machine (JVM).
D. Nei, Java krever spesifikk maskinvare for å fungere.

---

2. Kan du deklarere en variabel som int og senere redeklarere den som double i samme omfang (context)?

A. Ja, du kan deklarere variabelen på nytt som double.
B. Nei, en variabel kan ikke redeklareres med en annen type i samme omfang.
C. Bare hvis variabelen er deklarert som final.
D. Ja, men bare hvis du bruker nøkkelordet var.

---

3. Hva er integer overflow, og kan flyttallsoperasjoner forårsake overflow?

A. Overflow oppstår når en variabel får en verdi som er for stor til å lagres, og dette gjelder både heltalls- og flyttallsoperasjoner.
B. Overflow skjer kun når flyttallsoperasjoner overstiger lagringskapasiteten, og dette gjelder ikke heltallsoperasjoner.
C. Overflow oppstår når en variabel får en verdi som er for stor til å lagres, og dette gjelder ikke for noen av typene.
D. Overflow oppstår når en variabel får en verdi som er for stor til å lagres, men det gjelder bare heltallsoperasjoner. Flyttallsoperasjoner forårsaker ikke overflow.

---

4. Hva er en avrundingsfeil (round-off error), og hvilke typer operasjoner kan forårsake dette?

A. En avrundingsfeil oppstår i både heltalls- og flyttallsoperasjoner når det er en forskjell mellom en beregnet tilnærming av et tall og dets eksakte verdi.
B. En avrundingsfeil oppstår kun i heltallsoperasjoner fordi heltall ikke kan representere desimalverdier nøyaktig.
C. En avrundingsfeil oppstår kun i flyttallsoperasjoner fordi flyttall har begrenset presisjon. Heltallsoperasjoner forårsaker ikke avrundingsfeil.
D. En avrundingsfeil oppstår hver gang en beregning med tall overstiger lagringskapasiteten for datatypen.

---

5. Hvilket av følgende utsagn beskriver forskjellene mellom syntaksfeil, kjøretidsfeil og logiske feil på riktig måte?

A. Syntaksfeil: Oppdages av kompilatoren.
Kjøretidsfeil: Oppstår under kjøringen av programmet.
Logiske feil: Gir feil resultater selv om programmet kjører.
B. Syntaksfeil: Gir feil resultater selv om programmet kjører.
Kjøretidsfeil: Oppdages av kompilatoren.
Logiske feil: Oppstår under kjøringen av programmet.
C. Syntaksfeil: Oppstår under kjøringen av programmet.
Kjøretidsfeil: Gir feil resultater selv om programmet kjører.
Logiske feil: Oppdages av kompilatoren.
D. Syntaksfeil: Oppdages under kjøringen av programmet.
Kjøretidsfeil: Oppdages av kompilatoren.
Logiske feil: Forhindrer kompilatoren i å bygge programmet.

---

6. Hvilket av følgende utsagn er korrekt angående switch-setninger?

A. switch-variabler kan være av hvilken som helst datatype, og hvis break ikke brukes, avsluttes programmet umiddelbart.
B. switch-variabler kan være av char, byte, short, int, eller String-typer. Hvis break ikke brukes, utføres neste case-setning. Du kan alltid konvertere en switch-setning til en tilsvarende if-setning, men ikke omvendt.
C. switch-variabler kan bare være av int eller String-typer. Hvis break ikke brukes, avsluttes programmet med en feil. Du kan ikke konvertere en switch-setning til en if-setning.
D. switch-setninger er alltid mer effektive enn if-setninger og kan alltid brukes i stedet for en if-setning.

---

7. Hvilke formateringsspesifikatorer brukes for å skrive ut en boolsk verdi, et tegn, et heltall, et flyttall, og en streng i Java?

A. %b for boolean, %c for character, %d for decimal integer, %f for floating-point number, og %s for string.
B. %bool for boolean, %char for character, %int for decimal integer, %float for floating-point number, og %string for string.
C. %B for boolean, %C for character, %D for decimal integer, %F for floating-point number, og %S for string.
D. %bool for boolean, %c for character, %d for decimal integer, %fp for floating-point number, og %str for string.

---

8. Hvilket av følgende utsagn er korrekt om å konvertere en for-løkke til en while-løkke og fordelene med for-løkker?

A. Det er ikke mulig å konvertere en for-løkke til en while-løkke. for-løkker gir ingen spesielle fordeler sammenlignet med while-løkker.
B. Du kan alltid konvertere en for-løkke til en while-løkke. for-løkker hjelper med å unngå feil som å glemme å oppdatere kontrollvariabelen, for eksempel i++.
C. Du kan bare konvertere en for-løkke til en while-løkke hvis løkken ikke har en kontrollvariabel. for-løkker er mer effektive enn while-løkker.
D. for-løkker kan kun brukes til iterasjoner med faste grenser, mens while-løkker brukes til dynamiske iterasjoner.

---

9. Hvilket av følgende utsagn beskriver fordelene ved å bruke metoder?

A. Metoder lar deg skrive mindre kode, gjør programmet raskere, og eliminerer behovet for feilsøking.
B. Metoder gjør det mulig å gjenbruke kode, redusere kompleksitet, og forbedre ytelsen dramatisk.
C. Metoder gjør det mulig å gjenbruke kode, redusere kompleksitet, og gjøre programmet enklere å vedlikeholde.
D. Metoder tillater deg å skrive all logikk i én funksjon, noe som gjør det lettere å feilsøke.

---

10. Hvilket av følgende utsagn om argumenter og parametere er korrekt?

A. Argumenter sendes til metoder uten krav til rekkefølge eller type, og de kan ikke ha samme navn som parametrene.
B. Argumenter sendes til metoder ved å matche typen og rekkefølgen, og argumentet kan ha samme navn som parameteren.
C. Argumenter må alltid ha forskjellige navn fra parameterne, men rekkefølgen og typen er ikke viktig.
D. Argumenter sendes kun ved verdipassing, og de må alltid ha samme navn som parameteren.

---

11. Hvilket av følgende utsagn om metodeoverlasting er korrekt?

A. Metodeoverlasting tillater metoder med samme navn og parameterliste, men forskjellige returtyper eller modifikatorer.
B. Metodeoverlasting betyr å ha to eller flere metoder med samme navn og parameterliste i samme klasse.
C. Metodeoverlasting tillater metoder med samme navn, men med forskjellige parameterlister. Returtypen eller modifikatorene er ikke relevant for overlasting.
D. Metodeoverlasting betyr at metoder med samme navn må ha identiske parameterlister og modifikatorer, men kan ha ulike returtyper.

---

12. Hvilket av følgende utsagn beskriver forskjellen mellom en klasse og et objekt?

A. En klasse er en forekomst av et objekt, mens et objekt er en blåkopi for å lage klasser.
B. En klasse er en mal eller blåkopi som blant annet definerer egenskaper og metoder, mens et objekt er en konkret forekomst av klassen.
C. En klasse er en fysisk enhet i minnet, mens et objekt er en logisk abstraksjon.
D. Klasser og objekter er det samme i Java; det er ingen forskjell mellom dem.

---

13. Hvilket av følgende utsagn beskriver forskjellen mellom konstruktører og metoder?

A. Konstruktører er spesielle metoder som ikke kan ha parametere, mens metoder alltid har en returtype.
B. Konstruktører har alltid en returtype, men metoder trenger ikke å ha en returtype.
C. Konstruktører er spesielle metoder som kalles når et objekt opprettes med new, og de har ingen returtype – ikke engang void.
D. Konstruktører og metoder er det samme; forskjellen er bare navnekonvensjon.

---

14. Hvilket av følgende utsagn om arrayer er korrekt?

A. En array er en primitiv datatype. Den kan inneholde elementer av objekttyper, og standardverdien for elementer er null for alle typer.
B. En array er et objekt. Den kan ikke inneholde elementer av objekttyper, og standardverdien for alle typer er 0.
C. En array er et objekt. Den kan inneholde elementer av objekttyper, og standardverdien for elementer avhenger av typen: 0 for numeriske verdier, false for boolean, '\u0000' for char, og null for objekttyper.
D. En array er et objekt. Den kan inneholde elementer av objekttyper, men krever eksplisitt initialisering for alle elementer.

---

15. Hvordan beskrives et anonymt objekt i Java?

A. Et objekt som ikke har noe navn, men kan refereres til gjennom en statisk metode.
B. Et objekt som ikke har en referansevariabel som peker til det.
C. Et objekt som alltid opprettes i en konstruktør og aldri brukes utenfor klassen.
D. Et objekt som er erklært som null ved opprettelse.

---

16. Hvilket av følgende utsagn beskriver accessor- og mutator-metoder korrekt, inkludert navnekonvensjoner?

A. Accessor-metoder brukes for å hente private dataverdier, mens mutator-metoder brukes for å endre private dataverdier. Navnekonvensjonen er getDataFieldName() for ikke-boolean verdier og isDataFieldName() for boolean verdier for accessor-metoder, og setDataFieldName(value) for mutator-metoder.
B. Accessor-metoder brukes for å endre private dataverdier, mens mutator-metoder brukes for å hente private dataverdier. Navnekonvensjonen er getDataFieldName() for accessor-metoder og setDataFieldName(value) for mutator-metoder.
C. Accessor- og mutator-metoder kan brukes til både å hente og endre private dataverdier. Navnekonvensjonen er accessDataFieldName() for accessor-metoder og mutateDataFieldName(value) for mutator-metoder.
D. Accessor- og mutator-metoder brukes utelukkende til boolean dataverdier. Navnekonvensjonen er isDataFieldName() for begge metodene.

---

17. Hva er innkapsling (encapsulation) i Java?

A. En mekanisme som kombinerer data og metoder som opererer på dataene, og skjuler dataene fra direkte tilgang.
B. En prosess der alle metoder og data er gjort offentlig tilgjengelige.
C. En teknikk for å gjøre koden tilgjengelig for andre uten restriksjoner.
D. En mekanisme for å utføre flere operasjoner samtidig.

---

18. Hva er abstraksjon (abstraction) i Java?

A. Abstraksjon er en prosess som skjuler implementasjonsdetaljene til en klasse eller metode, og eksponerer kun dens essensielle funksjoner for brukeren.
B. En prosess der alle datafelter og metoder er offentlig tilgjengelige.
C. En teknikk for å kombinere flere objekter i én klasse.
D. En mekanisme for å lage objekter uten å deklarere noen metoder.

---

19. Hvilket av følgende utsagn beskriver korrekt bruk av super og forskjellen mellom private og protected tilgangsmodifikatorer?

A. super brukes til å erklære en klasse som en superklasse for alle andre klasser.
   private gir tilgang i samme klasse og subklasser, mens protected gir tilgang bare i samme klasse.
B. super brukes til å få tilgang til en superklasses metoder, konstruktører, eller felt.
   private gir tilgang kun i samme klasse, mens protected gir tilgang i samme klasse, subklasser, og andre klasser i samme pakke.
C. super lar en subklasse skjule metoder i en superklasse.
   private gir tilgang kun i samme pakke, mens protected gir tilgang kun i samme klasse og subklasser.
D. super brukes til å få tilgang til private medlemmer i superklassen.
   private og protected er identiske og gir tilgang kun i samme klasse.

---

20. Hva er fordelen med å bruke unntakshåndtering (exception handling) i Java?

A. Unntakshåndtering gjør det mulig for et program å ignorere feil og fortsette uten noen konsekvenser.
B. Unntakshåndtering gjør det unødvendig å håndtere feil i det hele tatt, da programmet automatisk løser alle feil.
C. Unntakshåndtering gjør det mulig for en metode å kaste et unntak til sin kallende metode, slik at unntaket kan håndteres der. Dette forhindrer at den kalte metoden må håndtere unntaket selv eller avslutte programmet.
D. Unntakshåndtering sikrer at programmet aldri vil avsluttes på grunn av en feil, uansett alvorlighetsgrad.

[Deloppgave 1. på norsk bokmål slutt.]

Del 2 – Informasjon

Denne delen består av 40 flervalg-oppgaver som hver gir 1 poeng (1 %) ved riktig svar, og -0,25 poeng i minus ved feil svar (total minus poeng som kan oppnås er -10). Du skal velge kun *ett* alternativ. Det kan være at ett alternativ er mer korrekt enn de andre alternativene, og det mest korrekte alternativet vil gi poeng. Hvis kandidaten foreslår mer enn ett riktig svar, så vil oppgaven ikke godkjennes.

Du kan besvare oppgaven i WISEflow ved å bruke FLOWmulti (som har støtte for flervalgsspørsmål), *eller*

skrive opp en liste med: *oppgavenummer+korrekt løsningsforslag: a,b,c eller d*,

f.eks. 1.b, 2.c, 3.a, osv. til og med oppgavenummer 40.

Del 2 – Flervalg om Java-kodeforståelse (40 %)

OPPGAVER

1. Anta at x er 0. Hva er utskriften av følgende kode?

```
if (x > 0)
    System.out.print("x er større enn 0");
else if (x < 0)
    System.out.print("x er mindre enn 0");
else
    System.out.print("x er lik 0");
```

A. x er mindre enn 0
B. x er større enn 0
C. x er lik 0
D. Ingen

---

2. Hva er verdien av y etter følgende switch-setning?

```
int x = 0;
int y = 0;
switch (x + 1)  {
    case 0: y = 0;
    case 1: y = 1;
    default: y = -1;
}
```

A. 2
B. 1
C. 0
D. -1

---

3. Analyser følgende kode:

Kode 1:
```
boolean even;

if (number % 2 == 0)
    even = true;
else
    even = false;
```

Kode 2:
```
boolean even = (number % 2 == 0);
```

A. Kode 2 har syntaksfeil.
B. Kode 1 har syntaksfeil.
C. Begge Kode 1 og Kode 2 har syntaksfeil.
D. Begge Kode 1 og Kode 2 er korrekte.

---

4. Hva er utskriften av dette programmet?

```
public class Test {
    public static void main(String[] args) {
        int x = 5;

        if (x > 0) {
            if (x < 10) {
                System.out.println("A");
            } else {
                System.out.println("B");
            }
        } else {
            System.out.println("C");
        }
    }
}
```

A. A
B. B
C. C
D. Ingen utskrift

---

5. Hva er verdien av x etter å ha evaluert følgende uttrykk?

```
x = (2 > 3) ? 2 : 3;
```

A. 5
B. 2
C. 3
D. 4

---

6. Analyser følgende kode:

```
int x = 0;
if (x > 0);
{
    System.out.println("x");
}
```

A. Verdien av variabelen x skrives ut.
B. Symbolet x skrives ut to ganger.
C. Symbolet x skrives ut.
D. Ingenting skrives ut fordi x > 0 er false.

---

7. Hvilke rettelser må gjøres for at koden skal fungere korrekt?

```
 1 public class Test {
 2    public void main(string[] args) {
 3      double i = 50.0;
 4      double k = i + 50.0;
 5      double j = k + 1;
 6
 7      System.out.println("j er " + j + " og
 8        k er " + k);
 9    }
10 }
```

A. Legg til static i metoden på linje 2.
   Endre string til String på linje 2.
   Korriger strengens format slik at den ikke brytes over to linjer (linje 7-8).
B. Fjern public fra metoden på linje 2.
   Endre string til String på linje 2.
   Del utskriften på linje 7-8 i to separate System.out.println()-setninger.
C. Endre void til Void på linje 2.
   Endre string til Integer på linje 2.
   Flytt strengene på linje 7-8 inn i én linje uten å endre koden ytterligere.
D. Endre string til String på linje 2.
   Fjern void fra metoden på linje 2.
   Behold strengene fordelt over to linjer, fordi det fungerer uansett.

---

8. Hvordan evalueres uttrykket count < 100 på følgende punkter i koden?

1. **Point A (rett etter at betingelsen while (count < 100) er sjekket).**
2. **Point B (etter count er inkrementert inne i løkken).**
3. **Point C (etter at løkken er ferdig).**

```
int count = 0;
while (count < 100) {
  // Point A
  System.out.println("Welcome to Java!");
  count++;
  // Point B
}
// Point C
```

A. På Point A: Alltid usant
   På Point B: Alltid sant
   På Point C: Noen ganger sant og noen ganger usant

B. På Point A: Noen ganger sant og noen ganger usant
   På Point B: Alltid sant
   På Point C: Alltid usant

C. På Point A: Alltid sant
   På Point B: Noen ganger sant og noen ganger usant
   På Point C: Alltid usant

D. På Point A: Alltid sant
   På Point B: Alltid sant
   På Point C: Noen ganger sant og noen ganger usant

---

9. Hva vil programmet skrive ut når det kjøres?

```
public class Test {
  public static void main(String[] args) {
    int i = 1;
    do {
      int num = 1;
      for (int j = 1; j <= i; j++) {
        System.out.print(num + "G");
        num += 2;
      }

      System.out.println();
      i++;
    } while (i <= 5);
  }
}
```

A. 1G
   1G2G
   1G2G3G
   1G2G3G4G
   1G2G3G4G5G

B. 1G
   3G
   5G
   7G
   9G

C. 1G
   1G3G
   1G3G5G
   1G3G5G7G
   1G3G5G7G

D. 1G
   1G3G
   1G3G5G
   1G3G5G7G
   1G3G5G7G9G

---

10. Hva er resultatet når koden kjøres?

```
int x = 1;
while (0 < x) && (x < 100)
  System.out.println(x++);
```

A. Løkken kjører uendelig.
B. Koden kompilerer ikke fordi (0 < x) && (x < 100) ikke er omsluttet av et par parenteser.
C. Koden kompilerer ikke fordi løkkens kropp ikke er omsluttet av klammer.
D. Tallene 1 til 99 skrives ut.

---

11. Hva er verdien av i etter at løkken er ferdig?

```
int y = 0;
for (int i = 0; i < 10; ++i) {
   y += i;
}
```

A. 10
B. undefined
C. 9
D. 11

---

12. Hva skjer når programmet kjøres?

```
public class Test {
   public static void main (String[] args) {
     int i = 0;
     for (i = 0; i < 10; i++);
       System.out.println(i + 4);
   }
}
```

A. Programmet kompilerer til tross for semikolon (;) på for-linjen, og skriver ut **4**.
B. Programmet kompilerer til tross for semikolon (;) på for-linjen, og skriver ut **14**.
C. Programmet har en kompilasjonsfeil på grunn av semikolon (;) på for-linjen.
D. Programmet har en kjøretidsfeil på grunn av semikolon (;) på for-linjen.

---

13. Hva vil bli returnert når metoden calculate(5) kalles?

```
public class Test {
    public static void main(String[] args) {
        System.out.println(calculate(5));
    }

    public static int calculate(int n) {
        if (n <= 1) {
            return 1;
        }
        int result = n * calculate(n - 2);
        if (result % 2 == 0) {
            result += 5;
        }
        return result;
    }
}
```

A. 75
B. 15
C. 115
D. 105

---

14. Hva returnerer følgende metodekall?

```
"abcdefgh".substring(1, 3)
```

A. abc
B. bcd
C. bc
D. cde

---

15. Hvilken av følgende tildelingssetninger er riktig for å tildele tegnet '5' til variabelen c?

A. char c = '5';
B. char c = 5;
C. char c = "5";
D. char c = "344";

---

16. Hvilken av følgende kodebiter vil føre til en kjøretidsfeil?

A. int[] array = {1, 2, 3};
   System.out.println(array[3]);
B. int x = 5 / 0;
C. String text = null;
   System.out.println(text.length());
D. Alle ovenfor

---

17. For å deklarere en konstant MAX_LENGTH inne i en metode med verdien 99.98, hva skriver du?

A. final double MAX_LENGTH = 99.98;
B. double MAX_LENGTH = 99.98;
C. final MAX_LENGTH = 99.98;
D. final float MAX_LENGTH = 99.98;

---

18. Hva vil programmet skrive ut?

```
import java.util.Arrays;

public class Test {
    public static void main(String[] args) {
        int[] array = {4, 2, 7, 1, 3};
        int result = findValueAfterSorting(array);
        System.out.println(result);
    }

    public static int findValueAfterSorting(int[] arr) {
        Arrays.sort(arr); // Sorterer matrisen i stigende rekkefølge
        return arr[arr.length / 2]; // Returnerer midtverdien
    }
}
```

A. 7
B. 4
C. 3
D. 2

---

19. Hvilket av de følgende boolske uttrykkene har feil syntaks?

A. (x > 0) || (x < 0)
B. (true) && (3 > 4)
C. !(x > 0) && (x > 0)
D. (x != 0) || (x = 0)

---

20. Hva skjer når programmet kjøres?

```
public class Test {
  public static void main(String[] args) {
    int[] x = new int[3];
    System.out.println("x[0] er " + x[0]);
  }
}
```

A. Programmet har en kompilasjonsfeil fordi størrelsen på arrayet ikke ble spesifisert ved deklarasjon av arrayet.
B. Programmet har en kjøretidsfeil fordi elementene i arrayet ikke er initialisert.
C. Programmet kjører fint og viser: `x[0] er 0`.
D. Programmet har en kjøretidsfeil fordi array-elementet x[0] ikke er definert.

---

21. Hva vil programmet skrive ut når det kjøres?

```
public class Test {
  public static void main(String[] args) {
    int list[] = {1, 2, 3, 4, 5, 6};

    for (int i = 1; i < list.length; i++)
      list[i] = list[i - 1];

    for (int i = 0; i < list.length; i++)
      System.out.print(list[i] + " ");
  }
}
```

A. 1 2 3 4 5 6
B. 2 3 4 5 6 6
C. 2 3 4 5 6 1
D. 1 1 1 1 1 1

---

22. Gitt følgende metodedefinisjoner:

Første metode: `public static double m(double x, double y)`
Andre metode: `public static double m(int x, double y)`

Spørsmål: Hvilken metode kalles for følgende utsagn?

```
a. double z = m(4, 5);
b. double z = m(4, 5.4);
c. double z = m(4.5, 5.4);
```

A. (a) kaller den **første** metoden, (b) kaller den **første** metoden, (c) kaller den **andre** metoden.
B. (a) kaller den **andre** metoden, (b) kaller den **andre** metoden, (c) kaller den **første** metoden.
C. (a) kaller den **andre** metoden, (b) kaller den **første** metoden, (c) kaller den **andre** metoden.
D. (a) kaller den **første** metoden, (b) kaller den **andre** metoden, (c) kaller den **første** metoden.

---

23. Hva vil programmet skrive ut når det kjøres?

```
class Test {
  public static void main(String[] args) {
    System.out.println(xMethod((double)5));
  }

  public static int xMethod(int n) {
    System.out.println("int");
    return n;
  }

  public static long xMethod(long n) {
    System.out.println("long");
    return n;
```

```
    }
}
```

A. Kompileringsfeil fordi ingen av metodene matcher argumenttypen.
B. long og 5
C. int og 5
D. Kompileringsfeil fordi metoden xMethod er definert med flere overbelastninger (overloads).

---

24. Hvilket av følgende utsagn om nøkkelordet `this` i Java er korrekt? Se eksempelkode under.

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public void printName() {
        System.out.println("Name: " + this.name);
    }

    public void setName(String name) {
        this.name = name;
    }

    public Person getThis() {
        return this;
    }
}
```

A. this kan brukes for å referere til nåværende objektet innenfor metoder og konstruktører.
B. this kan brukes for å referere til en hvilken som helst annen instans av samme klasse.
C. this kan brukes for å kalle private metoder i andre objekter av samme klasse.
D. this kan brukes som erstatning for super for å referere til en superklasse.

---

25. Hva vil programmet skrive ut?

Metoden er:

```
public static double m(double x) {
  x++;
  return 2 * x;
}
```

Koden som kaller metoden er:

```
int x = 1;
System.out.println(x + " " + m(x));
```

A. 1 2
B. 2 2
C. 1 4
D. 2 4

---

26. Hva skjer når programmet kjøres?

```
public class Test {
    public static void main(String[] args) {
        int[] x = new int[5];
        int i;
        for (i = 0; i < x.length; i++)
            x[i] = i;
        System.out.println(x[i]);
    }
}
```

A. Programmet har en kjøretidsfeil fordi den siste utskriften i main forårsaker en ArrayIndexOutOfBoundsException.
B. Programmet viser 4.
C. Programmet har en kompilasjonsfeil fordi i ikke er definert i den siste utskriften i main.
D. Programmet viser 0 1 2 3 4.

---

27. Hvilken return-setning er korrekt for denne metoden, og hvorfor er det riktig?

```
public static int[][] p()
```

A. return 1;
   Forklaring: Dette er riktig fordi int[][] er en todimensjonal representasjon, og en enkeltverdi kan tolkes som et todimensjonalt array.
B. return new int[][]{{1, 2, 3}, {2, 4, 5}};
   Forklaring: Dette er riktig fordi det oppretter en todimensjonal array som samsvarer med returtypen int[][].
C. return new int[]{1, 2, 3};
   Forklaring: Dette er riktig fordi en en-dimensjonal array kan tolkes som en todimensjonal array.
D. return {1, 2, 3};
   Forklaring: Dette er riktig fordi {...} kan brukes direkte som returverdi for arrays.

---

28. Hvilken av følgende syntakser brukes for at ClassA skal arve fra ClassB i Java?

A. class ClassA inherits ClassB
B. class ClassA extends ClassB
C. class ClassA : ClassB
D. class ClassA implements ClassB

---

29. Hvilket alternativ brukes for å definere en konstruktør i klassen Customer?

A. public Customer(String name) { }
B. public void Customer(String name) { }
C. public Constructor(String name) { }
D. public void Constructor(String name) { }

---

30. Hva blir den siste utskriftssetningen (System.out.println) i hovedmetoden?

```
public class Foo {
   static int i = 0;
   static int j = 0;

   public static void main(String[] args) {
     int i = 2;
     int k = 3;
     {
       int j = 3;
       System.out.println("i + j is " + i + j);
     }

     k = i + j;
     System.out.println("k is " + k);
     System.out.println("j is " + j);
   }
}
```

A. j is 3
B. j is 2
C. j is 1
D. j is 0

---

31. Hva vil System.out.println(list); skrive ut?

```
java.util.ArrayList<String> list = new java.util.ArrayList<>();
list.add("New York");
java.util.ArrayList<String> list1 =
(java.util.ArrayList<String>)(list.clone());
list.add("Atlanta");
list1.add("Dallas");
System.out.println(list);
```

A. [New York]
B. [New York, Atlanta]
C. [New York, Atlanta, Dallas]
D. [New York, Dallas]

---

32. Basert på koden nedenfor, hvilke metoder arver SubClass fra SuperClass?

```
class SuperClass {
    private void privateMethod() {
        System.out.println("Private method");
    }

    protected void protectedMethod() {
        System.out.println("Protected method");
    }

    public void publicMethod() {
        System.out.println("Public method");
    }
}

class SubClass extends SuperClass {
    public void testMethods() {
        // privateMethod(); // Ikke tilgjengelig: Private metoder
arves ikke
        protectedMethod(); // Tilgjengelig
        publicMethod();    // Tilgjengelig
    }
}

public class Test {
    public static void main(String[] args) {
        SubClass obj = new SubClass();
        obj.testMethods();
    }
}
```

A. Kun Public metoder.
SubClass arver bare de public metodene fra SuperClass, men ikke protected eller private metoder.

B. Protected og Public metoder.
SubClass arver de protected og public metodene fra SuperClass. Private metoder arves ikke.

C. Alle metoder: Private, Protected og Public.
SubClass arver alle metodene fra SuperClass, inkludert private metoder, men private metoder er skjulte og ikke direkte tilgjengelige.

D. SubClass arver ingen metoder, men kan bruke public og protected metoder via superklassen.
SubClass må bruke public og protected metoder via superklassen uten å direkte arve dem.

33. Hva vil programmet skrive ut når det kjøres?

```
public class C {
  public static void main(String[] args) {
    B[] o = {new A(), new B()};
    System.out.print(o[0].toString());
    System.out.print(o[1]);
  }
}

class A extends B {
  public String toString() {
    return "A" + super.toString();
  }
}

class B {
  public String toString() {
    return "B";
  }
}
```

A. AB
B. AAB
C. ABB
D. Ingen av de ovennevnte alternativer.

---

34. Hva vil programmet skrive ut når det kjøres?

```
interface A {
  void print();
}

class C {}

class B extends C implements A {
  public void print() { }
}

public class Test {
  public static void main(String[] args) {
    B b = new B();
    if (b instanceof A)
      System.out.println("b er en instans av A");
    if (b instanceof C)
      System.out.println("b er en instans av C");
  }
}
```

A. Ingenting.
B. b er en instans av A.
C. b er en instans av C.
D. b er en instans av A, etterfulgt av: b er en instans av C.

---

35. Hvilken av alternativene nedenfor kan overskrive denne metoden?

```
protected double xMethod(int x) {…};
```

A. private double xMethod(int x) {…}
B. protected int xMethod(double x) {…}
C. public double xMethod(double x) {…}
D. public double xMethod(int x) {…}

---

36. Hvilket alternativ er korrekt for objekter og arv?

Gitt følgende:

- **C er et grensesnitt (interface).**
- **B er en abstrakt klasse som delvis implementerer C.**
- **A er en konkret klasse som har en standardkonstruktør og som arver fra B.**

A. A a = new C();
B. A a = new B();
C. C b = new A();
D. B b = new B();

---

37. Hva vil programmet skrive ut når det kjøres?

```
class A {
  public A() {
    System.out.println(
      "Standardkonstruktøren for A blir kalt");
  }
}

class B extends A {
  public B(String s) {
    System.out.println(s);
  }
}

public class C {
  public static void main(String[] args) {
    B b = new B("Konstruktøren for B blir kalt");
  }
}
```

A. Ingen utskrift.
B. " Konstruktøren for B blir kalt".
C. " Standardkonstruktøren for A blir kalt" etterfulgt av " Konstruktøren for B blir kalt".
D. " Standardkonstruktøren for A blir kalt".

---

38. Hva skjer når denne koden kompileres eller kjøres?

```
public class Test1 {
  public Object max(Object o1, Object o2) {
    if ((Comparable)o1.compareTo(o2) >= 0) {
      return o1;
    }
    else {
      return o2;
    }
  }
}
```

A. Programmet har en syntaksfeil fordi Test1 ikke har en main-metode.
B. Programmet har en syntaksfeil fordi o1 er en Object-instans og ikke har en compareTo-metode.
C. Programmet har en syntaksfeil fordi du ikke kan caste en Object-instans som o1 til Comparable.
D. Programmet ville kompilere hvis ((Comparable)o1.compareTo(o2) >= 0) ble erstattet med (((Comparable)o1).compareTo(o2) >= 0).

---

39. Hvilken unntakstype kastes når dette programmet kjøres?

```
public class Test {
    public static void main(String[] args) {
        Object o = null;
        System.out.println(o.toString());
    }
}
```

A. ArrayIndexOutOfBoundsException
B. ClassCastException
C. NullPointerException
D. ArithmeticException

---

40. For å legge til data i en eksisterende fil, hvilken konstruktør brukes for å opprette en FileOutputStream for filen out.dat?

A. new FileOutputStream("out.dat")
B. new FileOutputStream("out.dat", false)
C. new FileOutputStream("out.dat", true)
D. new FileOutputStream(true, "out.dat")

[Deloppgave 2. på norsk bokmål slutt.]

Del 3 – Case (Java programmeringsoppgave/OOP-design) (40 %)

OPPGAVER

Del 3 – Informasjon

Eksamen består av to (2) caseoppgaver med deloppgaver hvor du selv skal formulere skriftlige svar i WISEflow. Hver deloppgave (markert som a), b), c) og d)) teller like mye.

Anbefaling: Les alle oppgaver før du begynner å svare, og planlegg tiden iht. vektleggingen. Sørg for å vise sensor bredden i din kunnskap, og bruk all tiden som er til rådighet. Gjør dine egne forutsetninger dersom du mener noe er uklart!

Selv om trykte og skrevne hjelpemidler formelt er tillatt, vil det forventes at tiden som er til rådighet benyttes på å bevare oppgavene. Det vil ikke telle positivt om det er direkte avskrift fra hjelpemidlene.

CASEOPPGAVE 1: Modellere en dyrepark i Java (25%)

Dette caset er designet for å teste dine ferdigheter i objektorientert programmering, inkludert bruk av abstrakte klasser, arv, polymorfi, exception handling, og datastrukturer som lister og mapper. Du skal bygge et system som simulerer en enkel dyrepark hvor ulike dyr har unike navn og kan lage lyder. Dyreparken håndterer en liste av dyr og gir mulighet til å legge til nye dyr, få dem til å lage lyder, og generere rapporter over hvilke dyr som finnes i parken.

Oppgavemål

- Grunnleggende OOP-konsepter: Opprette en abstrakt klasse og subklasser som demonstrerer arv og polymorfi.
- Feilhåndtering: Bruke exception handling for å sikre at feilinput blir håndtert korrekt.
- Arbeid med lister og mapper: Implementere en liste for å lagre objekter og bruke en map for å telle antall dyr av hver type.
- Systemtest: Bygge og teste en komplett applikasjon i main-metoden.

Scenarier som dekkes

- Opprettelse av dyr (f.eks. hunder, katter og fugler) med egendefinerte lyder.
- Legge til dyr i dyreparken, inkludert håndtering av feilinput (som dyr uten navn).

- Iterere gjennom alle dyrene for å «høre» (genere tekstobjekter) lydene de lager.
- Generere en rapport som viser antall dyr av hver type i parken.

Krav til systemet

- Systemet skal være robust mot feil, for eksempel dyr med tomme navn.
- Det skal være lett å utvide med nye dyretyper ved å legge til flere subklasser.

Oppgave 1 (25%):

Du skal modellere en enkel dyrepark i Java. Dyreparken inneholder ulike dyr, og hvert dyr har et navn og en unik lyd (lyden her skal være en tekststreng som simulerer lyden). Systemet skal også kunne generere en rapport over hvilke dyr som finnes i parken. Koden skal du skrive for hånd så godt du kan. Implementer bare det viktigste som oppgaven etterspør.

a) Opprett en abstrakt klasse Animal (2%)

Lag en abstrakt klasse Animal som har følgende egenskaper:

- En String-variabel name for å representere dyrets navn.
- En konstruktør som setter navnet.
- En abstrakt metode makeSound() som skal implementeres av subklasser.

b) Lag subklasser Dog, Cat, og Bird (2%)

Lag tre subklasser:

- Dog: Overstyr makeSound() slik at den returnerer "lyden"/strengen: "Bjeff".
- Cat: Overstyr makeSound() slik at den returnerer "lyden"/strengen: "Mjau".
- Bird: Overstyr makeSound() slik at den returnerer "lyden"/strengen: "Kvitre".

c) Lag en klasse Zoo for å administrere dyreparken (10%)

Implementer klassen Zoo med følgende funksjoner:

- En liste som lagrer objekter av typen Animal.
- En metode addAnimal(Animal animal) for å legge til et dyr i listen.
- Bruk exception handling for å håndtere feil, for eksempel hvis et dyr har et tomt navn eller null-verdi.

- En metode makeAllSounds() som itererer gjennom alle dyrene og får dem til å lage lyder (skrive ut tekstobjekter).

---

d) Utvid Zoo med en rapportfunksjon (6%)

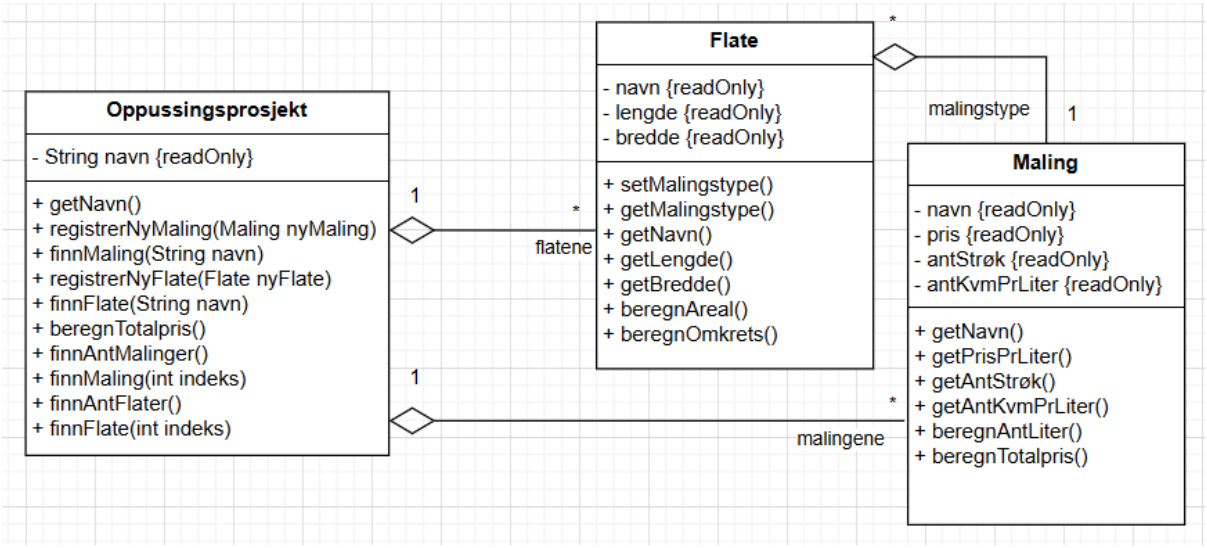
Legg til en metode generateReport() som teller hvor mange dyr det finnes av hver type og skriver dette ut som en rapport.

e) «Test» programmet i main-metoden (5%)

- Opprett flere objekter av Dog, Cat og Bird.
- Legg til dyrene i dyreparken, inkludert et tilfelle med feilinput.
- Kjør metoden makeAllSounds() for å få alle dyrene til å lage lyd (skrive ut tekstobjekter).
- Generer en rapport over antall dyr i dyreparken.

CASEOPPGAVE 2: Oppussing av en leilighet (15%)

Vi arbeider med et programsystem som beregner materialbehov og kostnader ved oppussing av en leilighet. Leiligheten består av mange rektangulære flater. Vi begrenser oss til én type oppussingsmateriale, nemlig maling. Vi skal programmere aggregater.



Beskrivelse:

I dette scenariet skal vi holde oversikt over malingsarbeidet i et oppussingsprosjekt. For hver flate registreres hvilken malingstype som skal brukes. Vi ønsker å vedlikeholde et register over alle flatene og et register over alle malingstypene, samt koble en malingstype til en flate etter behov.

Mellom klassene **Maling** og **Flate** etablerer vi en én-til-mange-relasjon: Én bestemt malingstype kan brukes på flere flater, men hver flate kan kun males med én type maling (i vårt eksempel).

For å håndtere dette samler vi begge registrene (flater og malingstyper) i et objekt av klassen **Oppussingsprosjekt**. Dette objektet tilbyr funksjonalitet som å registrere nye flater, legge til nye malingstyper, og hente informasjon om en spesifikk flate eller malingstype. Oppussingsprosjektet må derfor ha oversikt over hvilke flater og malingstyper som er registrert. I denne sammenhengen spiller objektene Flate og Maling viktige roller i prosjektet, med rollene **flatene** og **malingene**, henholdsvis (se UML-modellen).

Når det gjelder relasjonen mellom Flate og Maling, vurderer vi om en malingstype trenger å vite hvilke flater den er brukt på, og/eller om en flate trenger å vite hvilken malingstype som er brukt på den. Vi velger det siste: En flate må vite hvilken malingstype som er brukt på den, men en malingstype trenger ikke å kjenne til de enkelte flatene. Dette betyr at et malingsobjekt spiller rollen **malingstype** i relasjon til det tilhørende Flate-objektet.

Se vedlegg nederst i oppgaven:

- `Flate.java`
- `Maling.java`
- `Oppussingsprosjekt.java`
- `OppussingsKlient.java`

Oppgave 2 (15%):

Utvid klassen Oppussingsprosjekt med følgende metoder. Lag deretter et klientprogram som tester disse metodene, og svar på eventuelle spørsmål knyttet til implementasjonen. Du skal skrive koden for hånd så nøyaktig som mulig, og kun implementere det som er spesifikt beskrevet i oppgaven.

a) En metode for å fjerne en flate basert på navn. (6%)

b) En metode for å fjerne en malingstype. En malingstype kan bare fjernes dersom ingen flater har referanser til den. (6%)

c) Lag et eget klientprogram for å prøve ut disse metodene. Få samme utskrift som eksempelutskriften helt til slutt i vedlegget. (3%)

[Oppgavetekst slutt.]

Vedlegg:

```
/**
 * Flate.java
 *
 * Klassen Flate representerer en rektangulær flate med metoder for å
beregne
 * areal og omkrets, samt funksjonalitet for å identifisere flaten ved navn
og
 * å sette en malingstype.
 */
class Flate {
    private final String navn;  // Navn for identifikasjon av flaten
    private final double lengde; // Lengden av flaten (må være positiv)
    private final double bredde; // Bredden av flaten (må være positiv)
    private Maling malingstype;  // Type maling som er knyttet til flaten
(kan være null)

    /**
     * Konstruktør som oppretter en ny instans av Flate med et gitt navn,
lengde og bredde.
     * Kaster IllegalArgumentException hvis navn er null/blankt eller hvis
lengde/bredde er <= 0.
     *
     * @param navn Identifikasjonsnavn for flaten.
     * @param lengde Lengden på flaten i meter (må være positiv).
     * @param bredde Bredden på flaten i meter (må være positiv).
     * @throws IllegalArgumentException hvis navn er null eller tomt, eller
hvis lengde/bredde er <= 0.
     */
    public Flate(String navn, double lengde, double bredde) {
      if (navn == null || navn.trim().equals("")) {
        throw new IllegalArgumentException("Flatens navn må være
oppgitt."); // Sjekker at navn er gyldig
      }
      if (lengde <= 0.0 || bredde <= 0.0) {
```

```
    throw new IllegalArgumentException(
              "Både bredde og lengde må være positive tall.\n" +
              "Inndata til konstruktøren var: lengde = " + lengde +
              ", bredde = " + bredde);
  }
  this.navn = navn;
  this.lengde = lengde;
  this.bredde = bredde;
}

/**
 * Henter navnet på flaten.
 *
 * @return Navnet på flaten som en streng.
 */
public String getNavn() {
  return navn;
}

/**
 * Henter lengden av flaten.
 *
 * @return Lengden av flaten.
 */
public double getLengde() {
  return lengde;
}

/**
 * Henter bredden av flaten.
 *
 * @return Bredden av flaten.
 */
```

```
    public double getBredde() {
      return bredde;
    }

    /**
     * Beregner arealet av flaten som et produkt av lengde og bredde.
     *
     * @return Arealet av flaten.
     */
    public double beregnAreal() {
      return bredde * lengde; // Multipliserer lengde og bredde for å få
arealet
    }

    /**
     * Beregner omkretsen av flaten som summen av alle sidene.
     *
     * @return Omkretsen av flaten.
     */
    public double beregnOmkrets() {
      return 2.0 * (lengde + bredde); // Beregner omkrets ved å summere
lengde og bredde, og multiplisere med 2
    }

    /**
     * Setter malingstypen for flaten.
     *
     * @param nyMalingstype Ny type maling som knyttes til flaten.
     */
    public void setMalingstype(Maling nyMalingstype) {
      malingstype = nyMalingstype; // Oppdaterer malingstype med ny verdi
    }

    /**
```

```
     * Henter malingstypen som er knyttet til flaten.
     *
     * @return Malingstypen som er satt for flaten, eller null hvis ingen
malingstype er satt.
     */
    public Maling getMalingstype() {
      return malingstype;
    }


    /**
     * Sjekker om denne flaten er lik en annen flate basert på navnet.
     * Returnerer true hvis navnene er like, og false ellers.
     *
     * @param denAndre Objektet som sammenlignes med.
     * @return true hvis denAndre er en Flate med samme navn, false ellers.
     */
    public boolean equals(Object denAndre) {
      if (!(denAndre instanceof Flate)) {
        return false; // Returnerer raskt hvis objekttypen ikke er Flate
      }
      if (this == denAndre) {
        return true; // Returnerer true hvis objektene er identiske
      }
      Flate flate2 = (Flate) denAndre; // Typekonvertering til Flate
      return (navn.equals(flate2.navn)); // Sammenligner navnene for likhet
    }


    /**
     * Returnerer en tekstrepresentasjon av flaten, inkludert navn, bredde
og lengde.
     *
     * @return En streng som representerer flaten.
     */
    public String toString() {
```

```
      java.util.Formatter f = new java.util.Formatter();
      try {
      f.format("Flate: %s, bredde: %.2f m, lengde: %.2f m.", navn, bredde,
lengde); // Formaterer flateinformasjon
      return f.toString();
      } finally {
      f.close();  // Lukk formatteren for å unngå ressurslekkasje
    }
    }

/*
    /**
     * Testprogram for Flate-klassen. Utfører en enkel test av beregningene
for
     * areal og omkrets med forventede resultater og en toleranse.

    public static void main(String[] args) {
      final double TOLERANSE = 0.001; // Toleranse for å sammenligne
beregningsresultater
      System.out.println("Totalt antall tester: 2");

      // Oppretter et testobjekt av Flate med spesifikke dimensjoner
      Flate f1 = new Flate("A", 12.5, 7.3); // Forventet areal: 91,25 -
Forventet omkrets: 39,6

      // Tester beregning av areal mot forventet verdi med en liten
toleranse
      if (Math.abs(f1.beregnAreal() - 91.25) < TOLERANSE) {
        System.out.println("Flate: Test 1 vellykket");
      }

      // Tester beregning av omkrets mot forventet verdi med en liten
toleranse
      if (Math.abs(f1.beregnOmkrets() - 39.6) < TOLERANSE) {
        System.out.println("Flate: Test 2 vellykket");
      }
```

```
    }
*/
  }
```

```
/**
 * Maling.java
 *
 * Klassen Maling representerer en type maling og tilbyr metoder for å
beregne
 * materialbehov (mengde maling) og totalpris for å male en gitt flate.
 * Attributter: navn (identifikator), pris per liter, antall strøk og
dekkevne i kvm per liter.
 */
class Maling {
    private static final double GRENSE = 0.02;  // Toleransegrense for
avrunding til nærmeste 0.5 liter
    private final String navn;  // Navnet på malingen, brukes som
identifikator
    private final double pris;  // Pris per liter for malingen
    private final int antStrøk;  // Antall strøk som må males
    private final double antKvmPrLiter;  // Dekkevne i kvadratmeter per
liter


    /**
     * Konstruktør som oppretter en ny instans av Maling med de angitte
egenskapene.
     * Kaster IllegalArgumentException hvis navn er null/blankt eller hvis
pris, antStrøk
     * eller antKvmPrLiter er mindre enn eller lik null.
     *
     * @param navn Navnet på malingen (kan ikke være null eller blankt).
     * @param pris Prisen per liter maling (må være positiv).
     * @param antStrøk Antall strøk som kreves (må være positiv).
     * @param antKvmPrLiter Dekkevne per liter i kvadratmeter (må være
positiv).
     * @throws IllegalArgumentException hvis inndataene er ugyldige.
     */
```

```
    public Maling(String navn, double pris, int antStrøk, double
antKvmPrLiter) {
      if (navn == null || navn.trim().equals("")) {
        throw new IllegalArgumentException("Malingsnavn må være oppgitt.");
      }
      if (pris <= 0.0 || antStrøk <= 0 || antKvmPrLiter <= 0.0) {
        throw new IllegalArgumentException(
                  "Både pris, ant. strøk og dekkevne (kvm/l) må være
positive tall.\n" +
                  "Inndata til konstruktøren var: pris = " + pris +
                  ", ant. strøk = " + antStrøk + ", dekkevne (kvm/l) = " +
antKvmPrLiter);
      }
      this.navn = navn;
      this.antStrøk = antStrøk;
      this.antKvmPrLiter = antKvmPrLiter;
      this.pris = pris;
    }

    /**
     * Henter navnet på malingen.
     *
     * @return Navnet på malingen som en streng.
     */
    public String getNavn() {
      return navn;
    }

    /**
     * Henter prisen per liter maling.
     *
     * @return Prisen per liter.
     */
    public double getPrisPrLiter() {
      return pris;
```

```
    }

    /**
     * Henter antall strøk som er nødvendig.
     *
     * @return Antall strøk.
     */
    public int getAntStrøk() {
      return antStrøk;
    }

    /**
     * Henter dekkevnen til malingen, målt i kvadratmeter per liter.
     *
     * @return Dekkevne i kvadratmeter per liter.
     */
    public double getAntKvmPrLiter() {
      return antKvmPrLiter;
    }

    /**
     * Beregner antall liter maling som trengs for å male en gitt flate,
     * basert på flatearealet og antall strøk. Resultatet avrundes oppover
     * til nærmeste halve liter.
     *
     * @param enFlate Flate-objektet som skal males.
     * @return Antall liter maling som trengs, avrundet til nærmeste 0.5
liter.
     */
    public double beregnAntLiter(Flate enFlate) {
      double areal = enFlate.beregnAreal();  // Beregner arealet av flaten
      double antLiter = areal * antStrøk / antKvmPrLiter;  // Beregner
nødvendig liter maling
      int heltAntallLitere = (int) antLiter;  // Henter heltallsdelen av
literbehovet
```

```
      double overLiteren = antLiter - heltAntallLitere;  // Desimaldelen
(overflødig liter)


      // Avrunder oppover til nærmeste halve liter basert på
toleransegrensen GRENSE
      if (overLiteren >= 0.5 + GRENSE) {
        return heltAntallLitere + 1.0;
      } else {
        if (overLiteren >= GRENSE) {
          return heltAntallLitere + 0.5;
        } else {
          return heltAntallLitere;
        }
      }
    }


    /**
     * Beregner totalprisen for å male en gitt flate med denne malingen.
     *
     * @param enFlate Flate-objektet som skal males.
     * @return Totalpris for maling av flaten.
     */
    public double beregnTotalpris(Flate enFlate) {
      return beregnAntLiter(enFlate) * pris;  // Multipliserer nødvendig
liter med pris per liter
    }


    /**
     * To Maling-objekter anses som like hvis navnene deres er det samme.
     *
     * @param obj Objektet som sammenlignes med.
     * @return true hvis obj er av typen Maling og har samme navn; ellers
false.
     */
    public boolean equals(Object obj) {
```

```
      if (!(obj instanceof Maling)) {
        return false; // Returnerer raskt hvis objekttypen er ulik
      }
      if (this == obj) {
        return true; // Returnerer true hvis objektene er identiske
      }
      Maling m = (Maling) obj; // Typekonverterer til Maling
      return (navn.equals(m.getNavn())); // Sammenligner navnene for likhet
    }

    /**
     * Returnerer en strengrepresentasjon av malingsobjektet, inkludert
navn, pris, antall strøk
     * og dekkevne.
     *
     * @return Strengrepresentasjon av Maling-objektet.
     */
    public String toString() {
      java.util.Formatter f = new java.util.Formatter();
      try {
      f.format("Maling: %s, pris kr  %.2f  pr. liter, ant. strøk: %d, ant.
kvm/l %.2f",
                navn, pris, antStrøk, antKvmPrLiter); // Formaterer
malingsinformasjon
      return f.toString();
      } finally {
        f.close();  // Lukk formatteren for å unngå ressurslekkasje
      }
    }
  }
```

```
/**
 * Oppussingsprosjekt.java
 *
 * Klassen Oppussingsprosjekt vedlikeholder et register over flater og
malingstyper.
```

```
 * Klienten kan legge inn og hente ut objekter, og gjøre endringer i
mutable objekter.
 * For eksempel kan klienten hente ut et flateobjekt og endre malingstypen
for det.
 *
 * Klassene Maling og Flate er beskrevet utenfor denne filen.
 */


import java.util.ArrayList;


class Oppussingsprosjekt {
  private final String navn;  // Navnet på oppussingsprosjektet
  private ArrayList<Flate> flatene = new ArrayList<>();  // Liste over
registrerte flater
  private ArrayList<Maling> malingene = new ArrayList<>();  // Liste over
registrerte malingstyper


  /**
   * Konstruktør som oppretter et nytt Oppussingsprosjekt med gitt navn.
   *
   * @param navn Navnet på prosjektet.
   */
  public Oppussingsprosjekt(String navn) {
    this.navn = navn;
  }


  /**
   * Henter navnet på prosjektet.
   *
   * @return Navnet på prosjektet.
   */
  public String getNavn() {
    return navn;
  }
```

```
   /**
    * Registrerer en ny malingstype. Hvis en malingstype med samme navn
finnes,
    * returneres den eksisterende malingstypen. Ellers legges den nye til.
    *
    * @param nyMalingstype Malingstypen som skal registreres.
    * @return Den eksisterende eller nyregistrerte malingstypen.
    */
   public Maling registrerNyMaling(Maling nyMalingstype) {
     int indeks = malingene.indexOf(nyMalingstype);
     if (indeks < 0) {
       malingene.add(nyMalingstype);
       return nyMalingstype;
     } else {
       return malingene.get(indeks);
     }
   }

   /**
    * Søker etter en malingstype basert på navn.
    *
    * @param malingnavn Navnet på malingstypen som søkes etter.
    * @return Malingstypen med gitt navn, eller null hvis den ikke finnes.
    */
   public Maling finnMaling(String malingnavn) {
     for (Maling denne : malingene) {
       if (denne.getNavn().equals(malingnavn)) {
         return denne;
       }
     }
     return null;
   }

   /**
```

```
    * Registrerer en ny flate. Hvis en flate med samme navn finnes,
    * returneres den eksisterende flaten. Ellers legges den nye til.
    * Hvis flaten har en malingstype, registreres også denne om nødvendig.
    *
    * @param nyFlate Flaten som skal registreres.
    * @return Den eksisterende eller nyregistrerte flaten.
    */
   public Flate registrerNyFlate(Flate nyFlate) {
     int indeks = flatene.indexOf(nyFlate);
     if (indeks < 0) {
       flatene.add(nyFlate);
       // Hvis flaten har en malingstype som ikke er registrert,
registreres den nå
       Maling malingen = nyFlate.getMalingstype();
       if (malingen != null) {
         registrerNyMaling(malingen);
       }
       return nyFlate;
     } else {
       return flatene.get(indeks);
     }
   }

   /**
    * Søker etter en flate basert på navn.
    *
    * @param Flatenavn Navnet på flaten som søkes etter.
    * @return Flaten med gitt navn, eller null hvis den ikke finnes.
    */
   public Flate finnFlate(String Flatenavn) {
     for (Flate denne : flatene) {
       if (denne.getNavn().equals(Flatenavn)) {
         return denne;
       }
```

```
  }
  return null;
}

/**
 * Beregner totalprisen for alle flater i prosjektet basert på
 * deres areal og tilknyttede malingstype (hvis tilgjengelig).
 *
 * @return Totalprisen for alle malte flater i prosjektet.
 */
public double beregnTotalpris() {
  double totalpris = 0.0;
  for (Flate flaten : flatene) {
    Maling maling = flaten.getMalingstype();
    if (maling != null) {
      totalpris += maling.beregnTotalpris(flaten);
    }
  }
  return totalpris;
}

/**
 * Finner antall registrerte malingstyper i prosjektet.
 *
 * @return Antall malingstyper i prosjektet.
 */
public int finnAntMalinger() {
  return malingene.size();
}

/**
 * Henter en malingstype basert på indeks.
 *
```

```
    * @param indeks Indeksen for malingstypen.
    * @return Malingstypen på gitt indeks, eller null hvis indeksen er
ugyldig.
    */
   public Maling finnMaling(int indeks) {
     return (indeks >= 0 && indeks < malingene.size()) ?
malingene.get(indeks) : null;
   }

   /**
    * Finner antall registrerte flater i prosjektet.
    *
    * @return Antall flater i prosjektet.
    */
   public int finnAntFlater() {
     return flatene.size();
   }

   /**
    * Henter en flate basert på indeks.
    *
    * @param indeks Indeksen for flaten.
    * @return Flaten på gitt indeks, eller null hvis indeksen er ugyldig.
    */
   public Flate finnFlate(int indeks) {
     return (indeks >= 0 && indeks < flatene.size()) ? flatene.get(indeks)
: null;
   }

   /**
    * Lager en detaljert tekst som beskriver alle flater i prosjektet,
    * inkludert deres malingstyper, materialbehov og pris.
    *
    * @return En tekst med detaljer om flater og malingskostnader.
    */
```

```
  public String lagPrisOgBehovDetaljer() {
    java.util.Formatter f = new java.util.Formatter();
    double totalpris = 0.0;
    for (Flate flaten : flatene) {
      f.format(flaten.toString());  // Formaterer flateinformasjon
      Maling maling = flaten.getMalingstype();
      if (maling != null) {
        double behov = maling.beregnAntLiter(flaten);
        double pris = maling.beregnTotalpris(flaten);
        f.format(", behov: %.2f liter, pris %.2f kr.\n", behov, pris);
        totalpris += pris;
      } else {
        f.format(" Malingstype ikke bestemt\n");
      }
    }
    try {
    f.format("Totalpris %.2f kr.", totalpris);
    return f.toString();
   } finally {
       f.close();  // Lukk formatteren for å unngå ressurslekkasje
   }
  }

  /**
   * Returnerer en tekstlig representasjon av prosjektets flater og
malingstyper.
   *
   * @return En tekst som lister opp alle flater og malingstyper i
prosjektet.
   */
  public String toString() {
    String resultat = "Alle flater:\n";
    for (Flate enFlate : flatene) {
      resultat += enFlate + "\n";
```

```
    }
    resultat += "\nAlle malingstyper:\n";
    for (Maling enMaling : malingene) {
      resultat += enMaling + "\n";
    }
    return resultat;
  }
}
```

```
/**
 * OppussingsKlient.java
 *
 * Klienten tester ulike metoder i klassen Oppussingsprosjekt.
 * Demonstrerer registrering av malingstyper og flater, beregning av
totalpris,
 * og endring av malingstype for alle flater.
 */

class OppussingsKlient {
   public static void main(String[] args) {

      /* Registrerer to malingstyper og to flater */
      // Oppretter to malingstyper med navn, pris per liter, antall strøk
og dekkevne (kvm/l)
      Maling m1 = new Maling("Heimdal Super", 80, 2, 12);
      Maling m2 = new Maling("Heimdal Extra", 100, 3, 10);

      // Oppretter to flater med navn, lengde og bredde, og setter deres
malingstyper
      Flate f1 = new Flate("Vegg i barnerom", 4, 3);
      Flate f2 = new Flate("Taket i gangen", 6, 1);
      f1.setMalingstype(m1);  // Setter malingstype m1 på flate f1
      f2.setMalingstype(m2);  // Setter malingstype m2 på flate f2

      // Oppretter et nytt oppussingsprosjekt og registrerer malingene og
flatene
```

```
      Oppussingsprosjekt enLeilighet = new Oppussingsprosjekt("Min
leilighet");

      enLeilighet.registrerNyMaling(m1);

      enLeilighet.registrerNyMaling(m2);

      enLeilighet.registrerNyFlate(f1);

      enLeilighet.registrerNyFlate(f2);


      // Utskrift 1: Viser prosjektets flater og malingstyper etter
registrering

      System.out.println("Utskrift 1:\n" + enLeilighet.toString());


      /* Totalprisen for prosjektet */

      // Beregner og viser totalpris for alle flater i prosjektet, og
lukker formatter etter bruk

      try (java.util.Formatter f = new java.util.Formatter()) {

        f.format("Totalpris: %.2f",  enLeilighet.beregnTotalpris());

        System.out.println("Totalpris: " + f.toString());

      }


      /* Detaljert utskrift av flater med materialbehov og kostnad */

      System.out.println("\nDetaljer:\n" +
enLeilighet.lagPrisOgBehovDetaljer());


      /* Setter samme malingstype (m2) på alle flater */

      for (int i = 0; i < enLeilighet.finnAntFlater(); i++) {

        Flate flate = enLeilighet.finnFlate(i);  // Henter flate basert på
indeks

        flate.setMalingstype(m2);  // Setter malingstype m2 på flaten

      }


      // Utskrift 2: Viser detaljert oversikt over flater etter at
malingstypen er endret

      System.out.println("\nUtskrift 2:\n" +
enLeilighet.lagPrisOgBehovDetaljer());

    }

  }
```

```
  /* Forventet utskrift:

  Utskrift 1:

  Alle flater:

  Flate: Vegg i barnerom, bredde: 3,00 m, lengde: 4,00 m.

  Flate: Taket i gangen, bredde: 1,00 m, lengde: 6,00 m.


  Alle malingstyper:

  Maling: Heimdal Super, pris kr  80,00  pr. liter, ant. strøk: 2, ant.
kvm/l 12,00

  Maling: Heimdal Extra, pris kr  100,00  pr. liter, ant. strøk: 3, ant.
kvm/l 10,00


  Totalpris: Totalpris: 360,00


  Detaljer:

  Flate: Vegg i barnerom, bredde: 3,00 m, lengde: 4,00 m., behov: 2,00
liter, pris 160,00 kr.

  Flate: Taket i gangen, bredde: 1,00 m, lengde: 6,00 m., behov: 2,00
liter, pris 200,00 kr.

  Totalpris 360,00 kr.


  Utskrift 2:

  Flate: Vegg i barnerom, bredde: 3,00 m, lengde: 4,00 m., behov: 4,00
liter, pris 400,00 kr.

  Flate: Taket i gangen, bredde: 1,00 m, lengde: 6,00 m., behov: 2,00
liter, pris 200,00 kr.

  Totalpris 600,00 kr.

  */
```

Vedlegg slutt.