**UNIVERSITETET I OSLO**
**Det matematisk-naturvitenskapelige fakultet**

# Ny og utsatt skriftlig midtveiseksamen i IN1010
# 2026 vår
# Tid: 17. april kl 9:00 til 11:00

**Tillatte hjelpemidler:**
Ingen

**Annen informasjon**

Faglærerne kommer *ikke* til å besøke eksamenslokalet under eksamen. Om du mener at det er noe *alvorlig* galt med en oppgave, kan du ta kontakt med en eksamensvakt.

Alle oppgavene er uavhengig av hverandre og kan løses i vilkårlig rekkefølge.

Med mindre det står noe annet, er poenggivingen slik: Du får poeng for rett svar og du får 0 poeng for galt svar.

Det finnes en kalkulator tilgjengelig; se link nederst på skjermen.

**1** Hva skrives ut?
**Velg ett alternativ:**

- 32
- 0
- 16
- 1
- Kjørefeil
- 24 ✔

---

Maks poeng: 5

```java
class Arrayoppsett2 {
    public static void main (String[] args) {
        int[] a = new int[8];

        try {
            a[0] = 0;
            a[1] = 1;
            a[2] = 1;

            for (int ix = 3;  ix < a.length;  ix++) {
                a[ix] = a[ix-3] + a[ix-2] + a[ix-1];
            }

            System.out.println(a[7]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Kjørefeil");
        }
    }
}
```

**2** Hvilke tallverdier skrives ut når dette programmet kjøres?
**NB!** Legg merke til at innholdet i *Totalteller* er deklarert **static**.

1: ☐ (2) ☐ (1) ☐ (2)

2: ☐ (5) ☐ (2) ☐ (4)

3: ☐ (9) ☐ (3) ☐ (6)

4: ☐ (14) ☐ (4) ☐ (8)

---

Maks poeng: 12

```
class Totalteller {
    private static int antall = 0;
    public static void tell () {
        antall ++;
    }
    public static int hentAntall () {
        return antall;
    }
}

class Teller {
    private int antall = 0;
    public void tell () {
        antall ++;
        Totalteller.tell();
    }
    public void tell (int n) {
        antall += n;
        tell();
    }
    public int hentAntall (boolean lokal) {
        if (lokal) {
            return antall;
        } else {
            return Totalteller.hentAntall();
        }
    }
}

class TestTeller {
    public static void main (String[] args) {
        Teller minTeller = new Teller(),
               dinTeller = new Teller();

        for (int ix = 1;  ix <= 4;  ix++) {
            minTeller.tell(ix);
            dinTeller.tell();
            System.out.println(ix + ": " +
                minTeller.hentAntall(true) + " " +
                dinTeller.hentAntall(true) + " " +
                minTeller.hentAntall(false));
        }
    }
}
```

**3** Angi om det er arv (med klasser og subklasser) eller komposisjon som er best egnet til å implementere disse relasjonene.

(I denne oppgaven får du poeng for riktig svar, minuspoeng for galt svar og 0 poeng om du ikke svarer. Du vil imidlertid aldri få mindre enn 0 poeng totalt.)

**Velg hva som er best egnet:**

| | Komposisjon | Arv |
|---|---|---|
| Eksamen - prøveeksamen | ○ | ○ ✔ |
| Eksamen - eksamenslokale | ○ ✔ | ○ |
| Eksamen - midtveiseksamen | ○ | ○ ✔ |
| Innleveringsoppgave - KI-deklarasjon | ○ ✔ | ○ |
| Studieprogram - emne | ○ ✔ | ○ |
| Oppgave - flervalgsoppgave | ○ | ○ ✔ |
| Programmeringsspråk - Java | ○ | ○ ✔ |
| Eksamen - deloppgave | ○ ✔ | ○ |
| Metode - klasse | ○ ✔ | ○ |
| Datamaskin - laptop | ○ | ○ ✔ |

Maks poeng: 6

**4** Skriv de fire tallene som skrives ut:

☐ (16)

☐ (11)

☐ (8)

☐ (7)

---

Maks poeng: 8

```java
class ArrayBruk {
    public static void main (String[] args) {
        int[] alle = { 1, 3, 5 };
        int[] flere = new int[4];

        flere[3] = 7;
        int ind = 2;
        for (int t: alle) {
            flere[ind] = flere[ind+1] + t;
            ind --;
        }

        for (int u: flere)
             System.out.println(u);
    }
}
```

**5** I denne oppgaven skal du fylle ut alle blanke felt og sørge for at metoden *taUt* fjerner en oppgave med gitt tekst (String) fra den dobbeltkjedete listen av oppgaver.

Hint: *Oppgave*-objektene fungerer som *Node*-objekter i denne oppgaven.

Hjelp

```
abstract class Oppgave {
    private String beskrivelse ;
    Oppgave forrige, neste ;
    Oppgave ( [String beskr] ) { beskrivelse = beskr ; }
    public boolean navnEr ( [String beskr] ) {
        return beskrivelse . equals (beskr);
    }
}
class OppgaveListe  {
  Oppgave  første = null, siste = null ;
  Oppgave taUt ( [String beskr] ) {
    if ( første == null )  { return null; }
    Oppgave let = første ;
    while (let != siste && ! let . navnEr (beskr)) {
        [let = let.neste] ;
    }
    if ( ! let . navnEr(beskr))  { return null ; }
    if ( [let == første]  && let == siste)  { første = null ;  siste = null ; }
    // Mer enn ett element i listen:
    else if ( [let == første] ) { første = første . neste ; }
    else if ( [let == siste] ) { siste = siste . forrige ; }
    // Ta ut midt inne i listen:
    else  {
        let . forrige . neste = [let.neste] ;
        let . neste . forrige = [let.forrige] ;
    }
    return  let ;
 }
```

| | | | | |
|---|---|---|---|---|
| let == første | let == siste | let.forrige | let = let.neste | let |
| let = let.forrige | String beskr | let.neste | String let | Oppgave beskr |
| let == forrige | let == neste | | | |

Maks poeng: 12.96

**6** Hva skrives ut?

I denne oppgaven får du poeng for rett svar og minuspoeng for galt svar, men du vil aldri få færre enn 0 poeng totalt.

**Hva skrives ut?**

| | Mjau | Vov | GRR | pip | Vov-Vov | GRR-GRR | pip-pip |
|---|---|---|---|---|---|---|---|
| Arne : | ✔ | | | | | | |
| Bara : | | | | | | | ✔ |
| Cato : | | | | | | | ✔ |
| Dana : | | | | | | | ✔ |
| Dana (som Hund) : | | | | | | | ✔ |
| Elfi : | | | | | ✔ | | |
| Fogo : | | | | | ✔ | | |

Maks poeng: 9.03

```java
abstract class Dyr {
    abstract String lyd ();
}
class Katt extends Dyr {
    @Override String lyd () { return "Mjau"; }
}
class Hund extends Dyr {
    @Override String lyd () { return "Vov"; };
}
class Vakthund extends Hund {
    @Override String lyd () { return "GRR"; }
}
class Skjødehund extends Hund {
    @Override String lyd () { return "pip"; }
}
class Chihuahua extends Skjødehund {
    @Override String lyd () { return super.lyd() + "-" + super.lyd(); }
}
class Blindehund extends Hund { }
class BlindehundUnderOpplæring extends Blindehund {
    @Override String lyd () { return super.lyd() + "-" + super.lyd(); }
}

class MangeDyr {
    private static void giLyd (String navn, String lyd) {
        System.out.println(navn + ": " + lyd);
    }

    public static void main (String[] args) {
        Dyr arne = new Katt();
        Hund bara = new Chihuahua();
        Skjødehund cato = new Chihuahua();
        Chihuahua dana = (Chihuahua)bara;
        Hund elfi = new BlindehundUnderOpplæring();
        Blindehund fogo = (Blindehund)elfi;
        giLyd("Arne", arne.lyd());
        giLyd("Bara", bara.lyd());
        giLyd("Cato", cato.lyd());
        giLyd("Dana", dana.lyd());
        giLyd("Dana (som Hund)", ((Hund)dana).lyd());
        giLyd("Elfi", elfi.lyd());
        giLyd("Fogo", fogo.lyd());
    }
}
```

**7** Vi skal definere en klasse **Tabell10** som inneholder 10 String-er i en array. Klassen **Tabell10** skal få en iterator, men denne iteratoren skal gjennomløpe verdiene på en spesiell måte: Først skal elementene med partallsindeks (dvs 0, 2, 4, 6 og 8) hentes ut og deretter de med oddetallsindeks (dvs 1, 3, 5, 7 og 9).

Hjelp

```
class Tabell10 implements [ Iterable ✔ ] <String> {
   String verdier [ ] = new String [ 10 ] ;

   @Override public Iterator <String> [ iterator ✔ ] ( ) {
      return new [ Spesialiterator ✔ ] ( ) ;
   }
   class Spesialiterator implements [ Iterator ✔ ] <String> {
      int indeks = 0 ;
      boolean henterPartall = true ;
      @Override public boolean [ hasNext ✔ ] ( ) {
         return [ indeks < 10 ✔ ] ;
      }
      @Override public String [ next ✔ ] ( ) {
         String resultat = [ verdier [ indek ✔ ] ;
         indeks += 2 ;
         if ( [ indeks >= 10 ✔ ] && henterPartall ) {
            indeks = [ 1 ✔ ] ; henterPartall = false;
         }
         return [ resultat ✔ ] ;
      }
   }
}
```

| | | | |
|---|---|---|---|
| 1 | iterator | next | verdier [ indeks ] |
| indeks >= 10 | Iterator | indeks < 10 | hasNext |
| Spesialiterator | resultat | Iterable | |

Maks poeng: 11

**8** Denne oppgaven dreier seg om flybilletter. Det finnes ulike typer flybilletter som kan tilfredsstille alle eller noen av disse egenskapene:

- *KanOppgraderes* innebærer at kunden mot å betale litt ekstra kan få noen fordeler om bord.
- *KanRefunderes* innebærer at kunden kan få igjen den fulle billettprisen om han eller hun ikke reiser.
- *FåTilbakeSkatter* innebærer at kunden kan få refundert den delen av prisen som er avgifter (til staten og flyplassen).

Sett inn kodebitene på riktig plass slik at klassehierarkiet implementeres riktig.

Hjelp

```
interface KanOppgraderes {
   void oppgradering (int pris) ;
}
interface KanRefunderes [ extends ✔ ] KanOppgraderes {
   int hentRefusjonsverdi ( ) ;
}
interface FåTilbakeSkatter {
   int hentSkatter ( ) ;
}
class FlyBillett [ implements ✔ ] FåTilbakeSkatter {
    int pris;
    FlyBillett (int pris)  { this . [ pris ✔ ] = pris ; }
    @Override
    public int [ hentSkatter ✔ ] ( ) { return 100 ; } // Standard 100 kroner;
}
class FlyBillettFlex [ extends ✔ ] FlyBillett [ implements ✔ ] KanRefunderes {
   FlyBillettFlex (int pris)  { super (pris) ; }
   @Override
   public int [ hentRefusjonsve ✔ ] ( ) { [ return ✔ ] pris ; }
   @Override
   public void [ oppgradering ✔ ] ( int pris )  { this . pris += pris ; }
}
class FlyHalvFlex [ extends ✔ ] FlyBillett [ implements ✔ ] KanOppgraderes {
   FlyHalvFlex ( int pris )  { super (pris) ; }
   @Override
   public void [ oppgradering ✔ ] ( int pris ) { this . pris += pris ; }
}
```

| | | | |
|---|---|---|---|
| implements | 100 | hentSkatter | oppgradering |
| extends | pris | hentRefusjonsverdi | return |
| void | int | | |

Maks poeng: 12

**9** Metoden *settInn* bruker *compareTo* til å lete fra indeks 0 og utover i den sorterte arrayen etter det riktige stedet å sette inn bilen som er parameter. Når riktig sted er funnet, blir alle bilene på høyere indekser skjøvet en plass utover i arrayen. Fyll inn riktige kodebiter slik at metoden virker etter sin hensikt.

Hjelp

```
class Bilbeholder {
    Bil [ ] alleB = new Bil [100];
    private int antall = 0;
    public void settInn ( Bil nyBil ) {
        if ( antall == [alleB . leng ✔] ) { return ; } // ikke plass til flere
        boolean fortsett = [true ✔] ;
        int tell1 = 0 ;
        while ( [tell1 < anta ✔] && fortsett ) {
            if ( alleB [ tell1 ] . compareTo (nyBil) < 0 ) { fortsett = [false ✔] ; }
            else { [tell1 ++ ✔] ; }
        }
        for ( int tell2 = antall; tell2 > tell1; [tell2 -- ✔] ) {
            alleB [ [tell2 ✔] ] = alleB [ tell2 - 1 ] ;
        }
        alleB [ [tell1 ✔] ] = nyBil ;
        antall++ ;
    }
}
```

| | | | | |
|---|---|---|---|---|
| tell2 | tell1 -- | tell1 == antall | indeks | true |
| tell1 | tell2 ++ | tell1 ++ | false | antall |
| tell1 > antall | tell1 < antall | tell2 -- | alleB . length | |

Maks poeng: 13.04

**10** Hvilke tilordninger er tillatt (dvs gir verken kompileringsfeil eller kjørefeil) der det er markert ***/* Tilordning */*** ?

(I denne oppgaven vil du få poeng for rett svar, minuspoeng for galt svar og 0 poeng om du ikke svarer. Du vil imidlertid aldri få mindre enn 0 poeng totalt.)

**Angi om tilordning er tillatt eller ikke.**

| | Tillatt | Ikke tillatt |
|---|---|---|
| varB = (B) pA_C; | ✔ | |
| varC = (C) pA_C; | ✔ | |
| varV = (V) pA_C; | ✔ | |
| varA = pA; | ✔ | |
| varV = (V) pA; | | ✔ |
| varA = pA_B; | ✔ | |
| varB = pA_B; | | ✔ |
| varC = (B) pA_C; | | ✔ |
| varB = (C) pA_C; | ✔ | |
| varV = pA_C; | | ✔ |

Maks poeng: 11

```java
interface V { }
class A { }
class B extends A implements V { }
class C extends B { }

class Tilordning {
    public static void main (String[] args) {
        A pA = new A();
        A pA_B = new B();
        A pA_C = new C();

        A varA;
        B varB;
        C varC;
        V varV;

        /* Tilordning */
    }
}
```