Egenerklæring/ Declaration

Egenerklæring

Jeg erklærer herved at besvarelsen som jeg leverer er mitt eget arbeid.

Jeg har ikke:

- samarbeidet med andre studenter
- brukt andres arbeid uten at dette er oppgitt
- brukt eget tidligere arbeid (innleveringer/ eksamenssvar) uten at dette er oppgitt

Om jeg har benyttet litteratur *ut over pensum*, vil en litteraturliste inneholde alle kilder jeg har brukt i besvarelsen og referanser vil vise til denne listen.

Jeg er kjent med at brudd på disse bestemmelsene er å betrakte som fusk og kan føre til annullert eksamen og/eller utestengelse.

Dersom du er usikker på om du kan stille deg bak erklæringen, se retningslinjer for bruk av kilder i skriftlige arbeider ved Universitetet i Bergen og eventuelt ta kontakt med studieveileder/emneansvarlig.

Alle eksamensbesvarelser ved UiB blir sendt til manuell og elektronisk plagiatkontroll.

Merk: Ved å fortsette bekrefter jeg at jeg har lest erklæringen og at besvarelsen jeg leverer under denne eksamenen er mitt eget arbeid (og bare mitt eget arbeid), i full overensstemmelse med ovennevnte erklæringen.

Kontaktinfo under eksamen

For faglige spørsmål under eksamen
ta kontakt med foreleser via:

- epost adressert til: Martin.Vatshelle@uib.no
- I emnefeltet skriv: INF101 - eksamen.

NB: Følg med på INF101 V21 MittUiB siden . Om det dukker opp spørsmål / avklaringer i løpet av eksamen som er av interesse for flere studenter vil det bli lagt ut kunngjøring på MittUiB.

For IKKE-faglige spørsmål under eksamen (praktiske/ tekniske spørsmål om eksamen eller Inspera) ta kontakt med oss i studieadministrasjonen ved Institutt for Informatikk. Det er 2 forskjellige kanaler, e-post og telefon:

Når du tar kontakt med oss i studieadministrasjonen (enten per e-post eller telefon) ber vi deg å ha:

1. kandidatnummeret ditt tilgjengelig
2. studentnummeret ditt tilgjengelig
3. kontaktinfoen din dersom vi må henvise saken din videre (telefonnummer/ e-post).

Per e-post: studieveileder@ii.uib.no

1. I emnefeltet skriv: INF101 – eksamen
2. I selve e-posten trenger vi ditt kandidatnummer og studentnummer
3. Beskriv så kort som mulig hva problemet er

Per telefon i denne rekkefølgen:
55 58 41 82 – Mo Yan Yuen
55 58 41 59 – Eirik R. Thorsheim

For generelle eksamensinformasjon har fakultetet laget en infoside: https://www.uib.no/matnat/56756/eksamen-ved-det-matematisk-naturvitenskapelige-fakultet#eksamen-og-korona-nbsp-ofte-stilte-sp-rsm-l

Generelt info om denne eksamen

Denne eksamen består av 4 seksjoner

1. Informasjon (ingen oppgaver)
2. Flervalgsoppgaver (maks 20 poeng)
3. Programmering (maks 50 poeng)
4. Poeng fra Semesteroppgavene (maks 30 poeng)

På programmeringsdelen har vi på de fleste oppgavene laget litt ferdig kode som dere skal jobbe videre med.

Vi anbefaler at dere bruker en IDE på eksamen selv om dette ikke er påkrevd. Dere kan enten skrive direkte inn i Inspera eller kopiere fra editoren til Inspera når dere er ferdig med oppgaven. Har du endret en eksisterende fil et det tryggeste å kopiere hele filen inn i Inspera. Har du laget flere filer på en oppgave kopierer du alle filene etterhverandre i Inspera.

Det er mulig å kopiere all nødvendig kode fra Inspera til din editor, men for at dere skal spare tid har vi opprettet et GIT repository der dere kan laste ned all koden på en gang.
https://git.app.uib.no/ii/inf101/21v/students/exam_files

Når programmeringsoppgaver rettes ser vi på at det er gjort som oppgaven ber om og at det er brukt god kodestil. Det betyr gode variabel navn, gjenbruke av kode, lesbar kode, encapsulation osv.

1 Typeparameter

```
public class Grid<T extends ShoppingItem> {

    private List<T> cells;
    private int columns;
    private int rows;

    public Grid(int rows, int columns, T initElement) {
        if (rows <= 0 || columns <= 0)
            throw new IllegalArgumentException();
        this.columns = columns;
        this.rows = rows;
        cells = new ArrayList<>(columns * rows);
        for (int i = 0; i < columns * rows; ++i) {
            cells.add(initElement);
        }
    }
```

Hvilke datatyper kan denne Grid-klassen bruke (hvilke datatyper kan typeparameteren T være)?
Velg ett eller flere alternativer

- Alle datatyper som arver fra *Grid*
- *Grid* er generisk og kan dermed bruke alle datatyper
- Alle datatyper som er subtyper av *ShoppingItem*
- Kun primitive datatyper
- Kun imutable datatyper

Maks poeng: 5

2 Prinsipper i Objektorientert Programmering

```
public class Ball {

    private double radius;
    private Color color;

    public Ball(double radius, Color color) {
        this.radius = radius;
        this.color = color;
    }

    public Color getColor() {
        return color;
    }

    public double radius() {
        return radius;
    }

}
```

Hvilke av de følgende konseptene brukes i klassen *Ball*?
Velg ett eller flere alternativer

- Polymorphism
- Overriding
- Abstraction
- Encapsulation
- Generics
- Inheritance

Maks poeng: 5

3 Inheritance

I denne oppgaven tar vi for oss en kodebase som består av de følgende interfacene og klassene:

- et interface A
- et interface B som utvider A
- et interface C som utvider B
- en klasse D som implementerer A
- en klasse E som implementerer B
- en klasse F som implementerer C
- en klasse G som utvider D
- en klasse H som utvider E
- en klasse I som utivder F

I et program oppretter vi en variabel av typen B:

```
B minVariabel;
```

Hvilke typer kan et objekt ha for at det skal kunne legges i variabelen?

Med andre ord, hvilke typer kan en annen variabel `a` ha slik at følgende er lov:

```
minVariabel = a;
```

Hint: Det kan være nyttig å lage et klassediagram for disse klassene
Velg ett eller flere alternativer

- A
- B
- C
- D
- E
- F
- G
- H
- I

Maks poeng: 5

4 Typesignaturen til Collections.sort

For sortering av en liste i java kan en bruke metoden *Collections.sort* som har den følgende signaturen:

```
static <T extends Comparable<? super T>> void sort(List<T> list)
```

Velg de påstandene som er riktig for metoden *sort*.

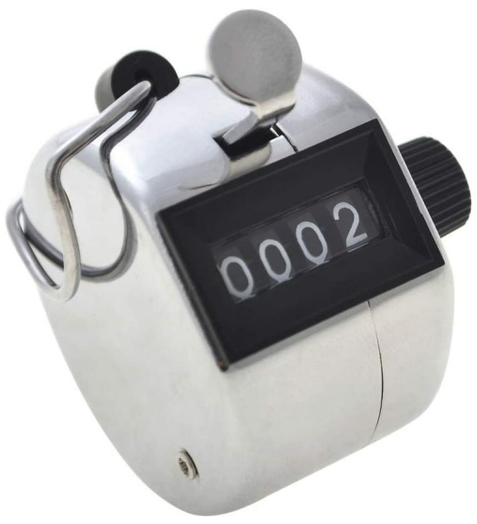
Velg ett eller flere alternativer

- "static" betyr at metoden kan kalles uten å først konstruere et Collections-objekt.
- Dersom en klasse som utvider T er sammenlignbar med seg selv, så kan List sorteres.
- Dersom T er sammenlignbar med seg selv, så kan List sorteres.
- "static" betyr at klassen List må være statisk for at en liste skal kunne sorteres.
- Klassen List må utvide klassen Comparable for at listen skal kunne sorteres.
- Dersom T utvider en klasse som er sammenlignbar med seg selv, så kan List sorteres.

Maks poeng: 5

5 Teller

Vi mennesker er dårlig til å telle, for eksempel antall personer som har gått inn i et lokale eller antall runder som har blitt løpt på banen. Vi prøver å oppdatere et tall i hodet mens vi teller, men dersom vi blir distrahert så glemmer vi fort hvor langt vi var kommet. I slike tilfeller er det nyttig å ha en *teller* som kan huske dette tallet for deg.



Kilde: https://www.amazon.com/GOGO-Counter-Carnival-Manual-Mechanical/dp/B001KX1VW2

En kan gjøre tre ting med en teller:

- lese av heltallet,
- øke tallet i displayet med 1
- og resette tallet til 0.

Telleren har også en datainvariant: Tallet kan aldri bli negativ.

Lag et interface *ICounter* som beskriver hva en teller kan gjøre, og lag en klasse *Counter* som implementerer *ICounter.* Sørg for at klassen overholder datainvarianten.
(Du trenger ikke å håndtere noen maksimum begrensning på telleren).

Skriv ditt svar her

```
1
```

Maks poeng: 8

6 ShoppingTest

Pål ønsker å utfordre Ebay med en handle-nettside. Så langt har han skrevet en handlevare-klasse *ShoppingItem*. Pål har også skrevet en test for om ulike handlevarer er like, men testen feiler.

Endre på *ShoppingItem* slik at testen passerer. *ShoppingItem*-objekter ansees å være like dersom de har samme type og samme merke (produsent).

```
@Test
public void shoppingItemEqualsTest() {
    ShoppingItem chicken1 = new ShoppingItem("chicken", "Prior");
    ShoppingItem chicken2 = new ShoppingItem("chicken", "Prior");
    ShoppingItem noodles = new ShoppingItem("noodles", "MrLee");

    assertNotEquals(chicken1, null);
    assertNotEquals(null, chicken1);

    assertNotEquals(chicken1, noodles);
    assertNotEquals(noodles, chicken1);

    assertEquals(chicken1, chicken1);
    assertEquals(chicken1, chicken2);
}
```

Skriv ditt svar her

```
1
```

Maks poeng: 8

7

Par

Et *par* er et objekt som inneholder to ordnede verdier, *first* og *second*, der de to verdiene kan være av ulike typer. Par kan blant annet brukes til å returnere to verdier fra en funksjon i stedet for kun en.

Denne oppgaven består av tre deler.

1. Lag et generisk interface *IPair* med to metoder *getFirst* og *getSecond* som henholdsvis henter det første og andre objektet i paret.

2. Lag et ikke-generisk interface *IIntegerPair* som representerer et par med to heltall. Interfacet skal utvide *IPair*, og skal ha en metode som returnerer summen av de to tallene.

3. Lag en klasse som implementerer *IIntegerPair* og har en konstruktør som tar inn de to tallene som argument. Man skal ikke kunne bytte ut verdiene i et eksisterende par (klassen skal med andre ord være immutable).

Skriv ditt svar her

```
1
```

Maks poeng: 10

8 IFridge

Pål har et sideprosjekt hvor han håper å utfordre Samsung i hvitevare-bransjen med et digitalt kjøleskap. Han har skrevet et interface for hvordan kjøleskapet skal fungere.

Implementer interfacet IFridge.

Skriv ditt svar her

```
1
```

Maks poeng: 10

9 Vaksineplan

Du utvikler en programvare som skal fordele vaksiner til pasienter. Hver pasient som skal vurderes har en alder og en *alvorlighetsgrad for underliggende sykdom*. Graden går fra 0, som betyr at pasienten ikke har en underliggende sykdom, til 3, som betyr at pasienten har en alvorlig underliggende sykdom.

Vaksinene kommer på ulike tidspunkt, og programmet må derfor sette opp en vaksineringskø. Hvor en pasient havner i køen avhenger av to (meget forenklede) faktorer:

- Dersom en pasient A har en høyere alvorlighetsgrad enn en pasient B, skal A komme før B i køen.
- Dersom alvorlighetsgraden er lik for to pasienter, skal den eldste av de to pasientene komme først i køen.

I denne oppgaven bruker vi en enum UnderlyingConditionGrade. For å for eksempel angi alvorlighetsgrad 1 kan en bruke `UnderlyingconditionGrade.LOW`. For å hente en alvorlighetsgrad fra et enum-objekt `u` kan en bruke `u.getValue()`.

Denne oppgaven består av tre deloppgaver. Det kan være lurt å gjøre seg kjent med klassene som hører til oppgaven før man går i gang med besvarelsen.

1. For å finne ut hvem som trenger vaksinen mest må vi kunne sammenlikne to pasienter. Bruk kriteriene ovenfor til å la Patient implementere Comparable<Patient> slik at den av to pasienter som trenger vaksinen **mest** kommer **før** den andre i en sortert liste.

2. Vaccine-klassen er en spesifikasjon av en generell vaksine. Utvid klassen med to klasser Pfizer og Moderna som henholdsvis representerer vaksinene "Pfizer" og "Moderna". Begge klassene skal ha en konstruktør som tar en LocalDate som argument og setter denne til å være leveringsdatoen.

3. Du skal nå fullføre main-metoden i VaccinePlan-klassen (du kan ta vekk tegnene som kommenterer ut klassen). I main-metoden har du en liste med pasienter og en liste med vaksiner, og du skal tildele vaksinene til pasientene slik at de som trenger vaksinene mest får først. Merk at rekkefølgen i de to listene er tilfeldige. Bruk hjelpemetoden assignVaccine for å tildele en vaksine til en pasient. (Dersom du ikke har fått til de forrige deloppgavene kan du likevel gjøre denne, selv om om du får noen feilmeldinger eller svaret blir feil.)

Tips: compareTo-metoden skal generelt returnere

- -1 dersom dette objektet (this) kommer før det andre,
- 1 dersom dette objektet (this) kommer etter det andre og
- 0 dersom dette objektet verken kommer før eller etter det andre.

Skriv inn all implementerte metoder og klasser her.

```
1
```

Maks poeng: 14

10 Poeng fra Semesteroppgavene V21

Denne oppgaven skal du ikke gjøre noe på. Her får du poeng du for det du har gjort på semesteroppgavene.

Du er nå ferdig med eksamen :-)

God Sommer
hilsen Martin

Skriv en hyggelig melding til foreleser ;-)

Maks poeng: 30

Question 6
Attached

```
@Test
public void shoppingItemEqualsTest() {
    ShoppingItem chicken1 = new ShoppingItem("chicken", "Prior");
    ShoppingItem chicken2 = new ShoppingItem("chicken", "Prior");
    ShoppingItem noodles = new ShoppingItem("noodles", "MrLee");

    assertNotEquals(chicken1, null);
    assertNotEquals(null, chicken1);

    assertNotEquals(chicken1, noodles);
    assertNotEquals(noodles, chicken1);

    assertEquals(chicken1, chicken1);
    assertEquals(chicken1, chicken2);
}
```

ShoppingItem:

```
/**
 * A shopping item is an item you purchase at the store. A shopping item has
    an
 * item type, such as chicken, fruit, vegetables, cleaning tools, soda, snacks
    ,
 * etc.,
 * and a brand, such as Fjordland, Tine, Prior, etc.
 *
 * @author Sondre Bolland
 *
 */
public class ShoppingItem {

    /**
     * Type of shopping item
     */
    private String itemType;
    /**
     * Brand (producer) of shopping item
     */
    private String brand;

    public ShoppingItem(String itemType, String brand) {
        this.itemType = itemType;
        this.brand = brand;
    }

    public String getItemType() {
        return itemType;
    }

    public String getBrand() {
        return brand;
    }
}
```

Question 8
Attached

IFridge:

```
import java.util.List;

public interface IFridge {

    /**
     * Returns the number of items currently in the fridge
     *
     * @return number of items in the fridge
     */
    int nItemsInFridge();

    /**
     * The fridge has a fixed (final) max size.
     * Returns the total number of items there is space for in the fridge
     *
     * @return total size of the fridge
     */
    int totalSize();

    /**
     * Place a food item in the fridge. Items can only be placed in the fridge
         if
     * there is space
     *
     * @param item to be placed
     * @return true if the item was placed in the fridge, false if not
     */
    boolean placeIn(FridgeItem item);

    /**
     * Remove item from fridge
     *
     * @param item to be removed
     * @throws IllegalArgumentException if fridge does not contain <code>item
         </code>
     */
    void takeOut(FridgeItem item);

    /**
     * Remove item from fridge.
     * Finds fridgeItem with <code>itemName</code>. If there is more than one
     * item with <code>itemName</code> find the item with oldest expiration
         date
     *
     * @param itemName
     * @throws IllegalArgumentException if fridge does not contain <code>item
         </code>
     * @return fridge item with oldest expiration date
     */
    FridgeItem takeOut(String itemName);

    /**
     * Remove all items from the fridge
     */
    void emptyFridge();
```

```
    /**
     * Remove all items that have expired from the fridge
     * @return a list of all expired items
     */
    List<FridgeItem> removeExpiredFood();

}
```

FridgeItem:

```
import java.time.LocalDate;

public class FridgeItem implements Comparable<FridgeItem> {

    private String name;
    private LocalDate expirationDate;

    public FridgeItem(String name, LocalDate expirationDate) {
        this.name = name;
        this.expirationDate = expirationDate;
    }

    /**
     * Checks the expiration date of the item against the current date.
     * If the expiration date is later than the current date then the item
     * has expired
     *
     * @return true if fridge item has expired
     */
    public boolean hasExpired() {
        return LocalDate.now().isAfter(expirationDate);
    }

    public String getName() {
        return name;
    }

    public LocalDate getExpirationDate() {
        return expirationDate;
    }

    @Override
    public String toString() {
        return name + ", " + expirationDate;
    }

    @Override
    public int compareTo(FridgeItem o) {
        return expirationDate.compareTo(o.expirationDate);
    }

}
```

Question 9
Attached

UnderlyingConditionGrade:

```
public enum UnderlyingConditionGrade {
    NONE(0), LOW(1), MEDIUM(2), HIGH(3);

    private int value;

    private UnderlyingConditionGrade(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}
```

Patient:

```
public class Patient {

    private String name;
    private UnderlyingConditionGrade underlyingConditionGrade;
    private int age;

    public Patient(String name, int age, UnderlyingConditionGrade
        underlyingCondition) {
        this.name = name;
        this.age = age;
        this.underlyingConditionGrade = underlyingCondition;
    }

    @Override
    public String toString() {
        return name;
    }
}
```

Vaccine:

```
public abstract class Vaccine implements Comparable<Vaccine> {

    private final LocalDate deliveryDate;

    public Vaccine(LocalDate deliveryDate) {
        this.deliveryDate = deliveryDate;
    }

    /**
     * @return the date for when the vaccine is delivered.
     */
    public LocalDate getDeliveryDate() {
        return deliveryDate;
    }

    /**
     * @return the name of the vaccine.
     */
    public abstract String getName();

    @Override
```

```
    public String toString() {
        return getName();
    }

    @Override
    public int compareTo(Vaccine o) {
        return this.deliveryDate.compareTo(o.deliveryDate);
    }
}
```

VaccinePlan:

```
public class VaccinePlan {

    public static void main(String[] args) {
        List<Patient> patients = getPatients();
        List<Vaccine> vaccines = getVaccines();

        // Fortsett her ...
    }

    private static void assignVaccine(Patient patient, Vaccine vaccine) {
        System.out.println(patient + " tildeles " + vaccine + "-vaksinen " +
            vaccine.getDeliveryDate() + ".");
    }


    private static List<Patient> getPatients() {
        return Arrays.asList(new Patient("Per", 43, UnderlyingConditionGrade.
            LOW),
                new Patient("Hans", 30, UnderlyingConditionGrade.NONE),
                new Patient("Anne", 51, UnderlyingConditionGrade.NONE),
                new Patient("Oddvar", 30, UnderlyingConditionGrade.HIGH),
                new Patient("Marie", 45, UnderlyingConditionGrade.NONE),
                new Patient("Gerd", 25, UnderlyingConditionGrade.MEDIUM));
    }

    private static List<Vaccine> getVaccines() {
        return Arrays.asList(
                new Pfizer(LocalDate.of(2021, 7, 1)),
                new Moderna(LocalDate.of(2021, 8, 5)),
                new Pfizer(LocalDate.of(2021, 8, 16)),
                new Moderna(LocalDate.of(2021, 8, 5)),
                new Pfizer(LocalDate.of(2021, 9, 3)),
                new Pfizer(LocalDate.of(2021, 8, 16)));
    }
}
```