

## 2 function1

Hvilket navn passer best for denne metoden?

```
public double function1(ArrayList<Number> input) {  
    double sum = 0.0;  
    for(Number number : input) {  
        sum += number.doubleValue();  
    }  
    return sum/input.size();  
}
```

Dokumentasjon for klassen Number kan finnes her:

<https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Number.html>

Velg et alternativ:

- ☐ toDouble
- ☐ sum
- ☐ doubleSum

☒ average



Riktig. 2 av 2 poeng.

## 3 Hvorfor Objekter

Hvorfor er det lurt å definere klasser og lage objekter?

Velg ett alternativ:

- ☐ Med flere klasser blir det mindre kode å skrive og dermed lettere å unngå feil (bugs) i koden.
- ☐ Jo flere klasser du har jo lettere blir det å forstå koden.
- ☐ Med flere klasser blir det mindre behov for dokumentasjon.

☒ Det viktigste med klasser er å organisere koden, de samler variabler og metoder som hører sammen slik at det blir lettere å finne frem i koden.



Riktig. 2 av 2 poeng.

## 4 Nyttig med testing

Det er viktig å teste koden din, til det bruker vi ofte JUnit tester. Det er lurt å tenke at de testene du skriver skal kunne brukes videre selv om deler av koden din endres i fremtiden.

For at koden din skal kunne testes med JUnit tester og at testene skal være nyttige selv når koden utvikler seg videre er følgende viktig når du skriver koden:

**Velg ett alternativ:**

☐ Skrive Javadoc kommentarer

☒ Bruke modularitet (modularity)



☐ Skrive lesbar kode

☐ Bruke GIT.

---


Riktig. 2 av 2 poeng.

## 5 Many Thanks

Her er 3 klasser, hva printes når main metoden kjøres?

```
public class Thanks {  
    public String toString(){  
        return "Thanks";  
    }  
}  
  
public class ManyThanks extends Thanks {  
    public String toString(){  
        return "Many "+super.toString();  
    }  
}  
  
public class SayThanks {  
    public static void main(String[] args) {  
        Thanks thank = new ManyThanks();  
        System.out.println(thank);  
    }  
}
```

Velg ett alternativ:

- ☒ Many Thanks 
- ☐ Vil få Exception fordi det ikke er implementert noen konstruktører.
- ☐ ManyThanks@2f92e0f4 (forskjellig tall hver gang man kjører programmet)
- ☐ Thanks

Riktig. 2 av 2 poeng.

## 6 function3

Jeg har prøvd å gå foran med et godt eksempel og skrevet god kode med utfyllende kommentarer på alle forelesningene. Når dere kommer ut i jobb vil mange av dere møte på noen som ikke tenker på lesbarhet når de programmerer :-)

Det må vi bare leve med og prøve å forstå hva koden gjør.

Her har noen skrevet en metode med dårlig metodenavn "function3" og ingen kommentarer. Det er heller ikke skrevet på den mest lettleselige måten.

Forstå hva koden gjør og velg det beste metodenavnet.

```
private static int function3(int n, int digit) {
    if(digit<0 || digit>9) {
        throw new IllegalArgumentException("This is not a valid digit");
    }
    if(n<0) {
        return function3(-n, digit);
    }
    if(n<10) {
        if(n==digit)
            return 1;
        else
            return 0;
    }

    int r = n%10;
    n = n/10;
    if(digit==r) {
        return function3(n,digit) + 1;
    }
    return function3(n,digit);
}
```

Velg ett alternativ:

- ☒ countMatchingDigits
- ☐ shiftDigits
- ☐ indexOfDigit
- ☐ digitSum (betyr tverrsum på norsk)

Riktig. 3 av 3 poeng.

## 7 Library

Denne enkle koden virker ikke som den skal. Meningen var å ha en liste over alle bøkene et bibliotek har og søke igjennom denne listen for å finne ut om biblioteket har en bestemt bok. Koden vår skriver ut "Oh no, I will not be an Java expert!". Hva er galt?

```
public class Library{
    ArrayList<Book> books = new ArrayList<Book>();

    Library(){
        //add books to the library
        books.add(new Book("Effective Java", "Joshua Bloch"));
        books.add(new Book("Clean Code", "Robert Martin"));
        books.add(new Book("Head First Design Patterns", "Elisabeth Robson"));
    }

    /**
     * Checks if the library has the book and prints a message
     * @param book
     */
    public void hasBook(Book book) {
        if(books.contains(book))
            System.out.println("Yes I can read this over summer!");
        else
            System.out.println("Oh no, I will not be an Java expert!");
    }

    public static void main(String[] args) {
        Library UiBLib = new Library();
        Book wanted = new Book("Clean Code", "Robert Martin");
        UiBLib.hasBook(wanted);
    }
}

public class Book {

    String title;
    String author;

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }
}
```

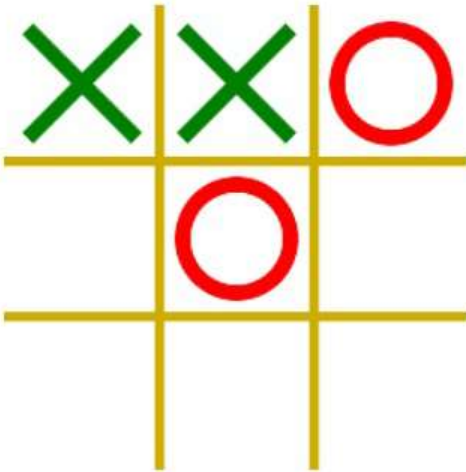
Velg ett alternativ:

- ☐ Book må implementere Comparable
- ☒ Vi må overlaste (override) equals metoden for å sjekke om to bøker er like.
- ☐ contains metoden fungerer bare på String og primitive variabler.
- ☐ Vi skulle hatt et ! (for negasjon) foran books.contains(book)

Riktig. 3 av 3 poeng.

## i Info Seksjon 2

Denne seksjonen handler om Semesteroppgave 2 og mer spesifikt er spørsmålene relatert til løsningsforslaget vi har laget til dere for noen uker siden og gikk igjennom med videoforelesninger.



Denne seksjonen tester om du har forstått løsningsforslaget til semesteroppgave 2 og om du har forstått noen grunnleggende konsepter innen objektorientert programmering.

Løsningsforslaget på semesteroppgave 2 finner dere her:

<https://retting.iib.no/inf101.v20.oppgaver/inf101.v20.sem2.losning>

<https://mitt.uib.no/courses/21613/files/2486903/download>

Skriv presise svar, hvis du klarer å trekke frem de rette punktene så er 5-10 linjer tekst per oppgave nok, du får ikke mer enn 4 poeng uansett om du skriver 3 sider.

## 8 AbstractPlayer

Se på løsningsforslaget til [Semesteroppgave 2](#). Der har vi valgt å implementere en klasse som heter AbstractPlayer. Hvorfor har vi gjort dette?

**Skriv ditt svar her**

AbstractPlayer implementerer interfacet Player og inneholder metoder som er felles for de forskjellige Player-typene. Dette er gjort for å samle metoder og variabler som er felles for alle forskjellige Player-typer i en klasse, slik at ConsolePlayer, GuiPlayer, RandomPlayer og MinMaxPlayer bare har metoder og variabler som er unike for disse klassene, mens de arver fellesmetodene fra AbstractPlayer. Slik unngår vi å skrive samme kode flere ganger, noe som letter feilsøking og oppdatering av kode.

---

Besvart.

## 9 Override

Se på løsningsforslaget til [Semesteroppgave 2](#). Beskriv et sted hvor vi overkjører (override) en metode som er implementert i superklassen og forklar hvorfor vi gjør dette.

**Skriv ditt svar her**

ConnectFour overrider `makeMove()` fra superklassen `AbstractGame` (og kaller deretter metoden den overrider med `super.makeMove()`). Dette gjøres siden `AbstractGame` håndterer spillregler som er felles for `TicTacToe` og `ConnectFour`, men i sistnevnte må brikken falle så langt ned på brettet som mulig før den plasseres. `ConnectFour.makeMove()` håndterer den forskjellen, før den kaller `super.makeMove()` med kvalitetssikret data og lar `AbstractGame` håndtere plassering av brikken uten hensyn til hvilken type spill vi spiller. Her har dere altså gjenbrukt koden i `AbstractGame.makeMove()` selv om den ble overridet i `ConnectFour`, så i stedet for å skrive hele metoden på nytt har dere bare endret en liten detalj. Dette gjør dere for å minimere duplikatkode, men likevel sikre at spillene oppfører seg forskjellig der de skal. Man kan også override hele metoden og ikke kalle superklassen i det hele tatt, igjen for å sikre at klassene oppfører seg forskjellig der de skal.

---

Besvart.

## 10 Polymorfisme

Se på løsningsforslaget til [Semesteroppgave 2](#). Beskriv et eksempel fra løsningsforslaget hvor det brukes polymorfisme. Beskriv både hvor i løsningsforslaget (hvilket metodekall) polymorfisme brukes og hvorfor det er polymorfisme.

Her er link til Oracle tutorials som forklarer [polymorfi](#).

**Skriv ditt svar her**

TicTacToe og ConnectFour er avarter av AbstractGame.

ConnectFour.makeMove() er et metodekall som bruker polymorfisme, og dette er polymorfisme siden det "endrer" AbstractGame til en annen klasse. Ref Oracle: "Subclasses of a class can define their own unique behaviors and yet share some of the same functionality of the parent class."

Derfor er også metodene TicTacToe.isWinner() og ConnectFour.isWinner() eksempler på polymorfisme, siden de har unik oppførsel og samtidig deler de funksjonalitet fra superklassen. Dette selv om de ikke overrider isWinner() fra superklassen (siden den ikke er med der, den er med i interfacet IGame).

---

Besvart.



## 11 Encapsulation

Se på løsningsforslaget til [Semesteroppgave 2](#). Beskriv et godt eksempel fra løsningsforslaget hvor innkapsling (encapsulation) er brukt og hvordan dette gjør koden bedre.

**Skriv ditt svar her**

AbstractGame bruker innkapsling på variablene gui og players. setGui() hindrer direkte tilgang til gui-variabelen. AbstractGame.addPlayer() er et bedre eksempel, her utføres det tester før en ny Player evt legges til i players-listen. Innkapsling gjør koden bedre ved å hindre at "hvem som helst" får tilgang til å endre variabler direkte. Dette er en fordel ved utvidning av koden, det kan være at de(n) opprinnelige utvikleren(e) har basert programvaren på en logikk som ikke er umiddelbart synlig, og ved å endre eksisterende variabler direkte i den nye utvidelsen kan man innføre uventede feil. Ved å innkapsle variabler og bruke getter- og setter-metoder, kan man gjøre variablene protected/private og man kan innføre sjekker i setter-metoden som ser om det man ønsker å legge til oppfyller ønskede krav. Man kan også la være å ha setter-metoder, slik at variablene bare kan leses.

---

Besvart.

## 12 locations

Se på løsningsforslaget til [Semesteroppgave 2](#). I klassen GameBoard i metoden count bruker vi en linje

```
for(Location loc : locations()) {
```

Forklar denne kodelinjen,

- hva gjør koden?
- hvilken kode er linjen avhengig av? List opp metoder/klasser i løsningsforslaget som blir kalt av denne linjen.

**Skriv ditt svar her**

(Denne linjen finner jeg i `countNumInRow()` og ikke i variantene av `count()`).

-Denne linjen inneholder en `foreach`-løkke, som går gjennom alle `Location`-instanser som returneres av `location()`. For hver iterasjon tilegner løkken en lokasjon fra `location()` variabelnavnet `loc`, og lar programmet utføre koden som følger fram til avsluttende krølleparantes. Dette gjøres for hver `Location` i `Iterable` som returneres av `locations()`.

-Denne linjen er først og fremst avhengig av en `Iterable`, som inneholder en liste av `Location` som `foreach`-løkken kan gå gjennom. Den får den fra `Grid.locations()`

Metoder og klasser som blir kalt av denne linjen:

- Klassen `Player`, input til `GameBoard.countNumInRow` er av typen `Player`
- Klassen `Location` (`foreach`-løkken sjekker at instansene fra `locations()` er av denne typen)
- Metoden `Grid.locations()` blir kalt, som igjen kaller
- Klassen `GridLocationIterator` (via konstruktøren som kalles i `Grid.locations()`)
- I bakgrunnen kaller deretter `foreach`-løkken metodene `GridLocationIterator.hasNext()` og `GridLocationIterator.next()` for å iterere gjennom listen av `Location`
- I hver iterasjon kalles `Math`-klassen og metoden `max()`.
- Inni kallet til `Math.max()` kalles metoden `GameBoard.count()`
- `GameBoard.count()` har en egen `foreach`-løkke, som er avhengig av enum-klassen `GridDirection`.

---

Besvart.

## i Info section 3

Denne seksjonen inneholder 4 oppgaver fordelt på 3 tema der du skal implementere kode.

- Alle oppgavene kommer med noe kode, to av dem kommer med interface du skal implementere og en kommer med ferdig kode som du skal teste og fikse.
- Dere velger selv om dere skriver rett inn i Inspira eller om dere programmerer i annen editor og kopierer inn svaret
- Interfacene og koden gitt i denne seksjonen er ferdig med javadoc, du trenger ikke skrive noen javadoc kommentarer.
- Det kan lønne seg å skrive noen kommentarer, særlig hvis du er usikker på om du har gjort rett.
- God kodelstil og lesbar kode gir poeng, gode kommentarer kan veie opp for at koden din ikke er så lesbar.
- Hvis du vet hva du skal gjøre men ikke får til å skrive koden kan du få noen poeng for å forklare hva du mente å gjøre.
- Dere blir bedømt både på at koden er korrekt og på kodekvalitet, e.g. variabelnavn.
- Vi trekker ikke poeng for små syntax feil som glemt semikolon på slutten av linjen.

## 13 ShoppingList

En handleliste er en liste med varer. Vi har implementert `ShoppingItem` som representerer varer og et interface `IShoppingList`. Du må skrive en klasse `ShoppingList` som implementerer `IShoppingList`. Kommentarene i `IShoppingList` hjelper deg å finne ut hva du må gjøre.

[IShoppingList.java](#)

[ShoppingItem.java](#)

1. Klassen må ha en konstruktør. Du velger selv hvilke input parametre denne konstruktøren skal ha, velg det som er lettest for deg. Du får ikke ekstra poeng for fancy konstruktører.
2. Det er 5 metoder i interfacet, implementer alle disse metodene.

## Skriv ditt svar her

```
1 package eksamen; // Put this in the appropriate package according to your file system
2
3 import java.time.LocalDate;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 public class ShoppingList implements IShoppingList {
8
9     private List<ShoppingItem> list;
10    private LocalDate createdDate;
11
12    public ShoppingList() {
13        list = new ArrayList<>();
14        createdDate = LocalDate.now();
15    }
16
17    @Override
18    public List<ShoppingItem> getAllItems() {
19        return list;
20    }
21
22    @Override
23    public void add(ShoppingItem item) {
24        if (item == null) {
25            throw new NullPointerException("ShoppingItem cannot be null");
26        }
27        else if (item.getName().equals("")) {
28            throw new IllegalArgumentException("The item needs a name");
29        }
30        else if (item.getPrice() < 0) {
31            throw new IllegalArgumentException("The items can not have negative prices (but they can
32                be free)");
33        }
34        else {
35            list.add(item);
36        }
37    }
38
39    @Override
40    public LocalDate getDate() {
41        return createdDate;
42    }
43
44    @Override
45    public double totalPrice() {
46        double total = 0;
47        for (ShoppingItem item : list) {
48            total += item.getPrice();
49        }
50        return total;
51    }
52
53    @Override
54    public int getItemCount(String itemName) {
55        int count = 0;
56        for (ShoppingItem item : list) {
57            if (item.getName().equalsIgnoreCase(itemName)) {
58                count++;
59            }
60        }
61        return count;
62    }
63 }
```

Besvart.

## 14 CoronaStats

Coronaviruset har gitt oss utfordringer på mange måter, men det gir også noen nye muligheter. Websider som [worldometers.info](https://worldometers.info) har fått mange nye besøkende som følger med på statistikken. For å få til en slik website ligger det en del programmering bak.

Vedlagt til denne oppgaven er et interface kallt ICoronaData som dere skal jobbe med i denne oppgaven.

[ICoronaData.java](#)

Dere skal implementere følgende to metoder: `cumulativeDeaths()` og `deathsPerMillion()`.

Enten lag en abstrakt en klasse som implementerer interfacet ICoronaData, eller implementer default metoder i ICoronaData.

Hvis du trenger å implementere andre metoder er det greit, men det er de to metodene nevnt over du får poeng for.

**Skriv ditt svar her...**

```
1 package eksamen; // Put this in the appropriate package according to your file system
2
3 import java.util.ArrayList;
4
5 public abstract class AbstractCoronaData implements ICoronaData {
6
7     public ArrayList<Integer> cumulativeDeaths() {
8         ArrayList<Integer> cumulative = new ArrayList<>();
9
10        for (Integer deaths : getDailyDeaths()) {
11            if (cumulative.size() == 0) {
12                cumulative.add(deaths);
13            }
14            else {
15                cumulative.add(cumulative.get(cumulative.size() - 1) + deaths);
16            }
17        }
18        return cumulative;
19    }
20
21    public ArrayList<Double> deathsPerMillion() {
22        ArrayList<Double> dailyDPM = new ArrayList<>();
23        Double population = Double.valueOf(getPopulation());
24        population /= Math.pow(10, 6);
25        for (Integer cumulativeDeaths : cumulativeDeaths()) {
26            Double deathsPerMillion = cumulativeDeaths / population;
27            dailyDPM.add(deathsPerMillion);
28        }
29        return dailyDPM;
30    }
31 }
```

Besvart.

## 15 ScrabbleTest

Spillet Scrabble skal lages. Spillet går ut på å omorganisere noen av de 7 gitte bokstavene til et ord. Vi må sjekke at det ordet spilleren ønsker å lage faktisk går an å lage med de bokstavene spilleren har. Vi skiller ikke på små og store bokstaver, vi ønsker å godta input som inneholder en blanding av små og store bokstaver og gjør om til store bokstaver inne i metoden. (For denne oppgaven antar vi at det kun brukes bokstaver fra det engelske alfabetet).

Scrabble har en blank brikke som brukes som 'wildcard' I denne oppgaven ser vi bort fra dette, men i INF102 vil dere lære det.

Vi er kommet godt i gang med spillet og har skrevet følgende metode med tilhørende JUnit tester. Denne metoden skal kun sjekke om det er mulig å lage ordet fra bokstavene, ikke om ordet er lov å legge ut (finnes i ordlisten og passer med de eksisterende ordene).

Vi ønsker at metoden fungerer uansett hvor lang "letters" og "word" er.

### [Scrabble.java](#)

```
/**
 * This method is used by the game Scrabble where the goal is to rearrange
 * some of the given letters into a word.
 * A letter in Scrabble is always upper case, this method accepts both upper and lower case letters
 * and considers e.g. a lower case 'a' equal to an upper case 'A'
 * You may assume that all letters in input are valid letters in the English alphabet.
 */
@param letters - the letters you have to your disposal
@param word - the word you are trying to form
@return true if it is possible to form the word by rearranging the letters, false otherwise
*/
public static boolean canMake(String letters, String word) {
    letters = letters.toUpperCase();
    word = word.toUpperCase();

    for(Character c:word.toCharArray()) {
        if(!letters.contains(c.toString()))
            return false;
    }
    return true;
}
```

Vi skrev to JUnit tester og begge testene passerer, og vi får ingen Exception når vi kjører spillet.

### [ScrabbleTest](#)

```
@Test
void testCanMakeEksamen(){
    assertTrue(Scrabble.canMake("SKAMENE", "EKSAMEN"));
    assertTrue(Scrabble.canMake("skaMENE", "EKSAMEN"));
    assertTrue(Scrabble.canMake("SKAMENE", "eksAMEN"));
}

@Test
void testCanNotMakeFerie(){
    assertFalse(Scrabble.canMake("SKAMENE", "FERIE"));
}
```

Men det virker ikke slik som vi forventet (en logisk feil gjør at vi av og til får lov til å legge ut ord som ikke går an å lage) og vi er sikker på at det er metoden canMake som er feil. Du må hjelpe med å finne feilen.

Finn feilen og skriv en test som fanger opp den feilen som er i koden, det vil si en test som feiler slik koden er nå, men vil passere når feilen er rettet opp.

**Skriv kode for test her**

1	@Test	
2	void testFailsBeforeFixPassesAfterFix() {	
3	// Feilen er at man tillater at bokstaver brukes flere ganger ved å ikke fjerne de brukte bokstavene fra letters. Bytter i denne testen ut den siste E med en A i SKAMENE slik at letters blir SKAMENA, testen passerer fortsatt selvom vi nå bare kan skrive AKSAMEN eller EKSAMAN	
4	assertFalse(Scrabble.canMake("SKAMENA", "EKSAMEN"));	
5		
6	}	

Besvart.

## 16 ScrabbleFix

Forrige oppgave (ScrabbleTest) var å skrive en test som feiler. Skriv om metoden canMake for å fikse feilen.  
**Skriv ditt svar her**

```
1 public static boolean canMake(String letters, String word) {  
2     letters = letters.toUpperCase();  
3     word = word.toUpperCase();  
4  
5     List<Character> lettersList = new ArrayList<>();  
6     for (Character c : letters.toCharArray()) {  
7         lettersList.add(c);  
8     }  
9  
10    for(Character c : word.toCharArray()) {  
11        if (lettersList.contains(c)) {  
12            lettersList.remove(c);  
13        }  
14        else {  
15            return false;  
16        }  
17    }  
18    return true;  
19 }
```

---

Besvart.

## 17 Poeng fra Semesteroppgavene

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!

Håper å se deg igjen i INF102.

Vennlig hilsen

Martin

**Skriv en hyggelig melding til foreleser ;-)**

God sommer :)

---

Besvart.