UNIVERSITETET I BERGEN

KANDIDAT

181

PRØVE

INF101 0 Objektorientert programmering

Emnekode	INF101
Vurderingsform	Skriftlig eksamen
Starttid	24.05.2024 09:00
Sluttid	24.05.2024 14:00
Sensurfrist	--
PDF opprettet	31.05.2024 13:17

Informasjon om Eksamen

Oppgave	Tittel	Oppgavetype
ℹ	Egenerklæring/ Declaration, INF101	Informasjon eller ressurser
ℹ	Info om eksamen V24	Informasjon eller ressurser

Konsepter - Avkryssing

Oppgave	Tittel	Oppgavetype
1	Box	Flervalg
2	Tetromino 1	Flervalg
3	Tetromino 2	Flervalg

Programmering og Teori

Oppgave	Tittel	Oppgavetype
4	Klasse VS Objekt	Langsvar
5	Calculator Polymorfisme	Langsvar
6	Calculator Tests	Flervalg (flere svar)
7	DNASequence	Programmering

Sudoku design avkryssing

Oppgave	Tittel	Oppgavetype
ℹ	SudokuBoard design info	Informasjon eller ressurser
8	SudokuBoard Arv vs Composition	Paring

9	SudokuBoard sizes	Paring
10	SudokuBoard Encapsulation	Paring
11	SudokuBoard and IGrid	Paring

Poeng fra Semesteroppgavene

Oppgave	Tittel	Oppgavetype
12	Poeng fra Semesteroppgavene V24	Tekstfelt

1 Box

Vi har opprettet en rekke klasser for å holde på noen verdier.

IntegerBox.java:

```java
public class IntegerBox {

    private int item;

    public IntegerBox(int item) {
        this.item = item;
    }
```

StringBox.java:

```java
public class StringBox {

    private String item;

    public StringBox(String item) {
        this.item = item;
    }
```

BooleanBox.java:

```java
public class BooleanBox {

    private boolean item;

    public BooleanBox(boolean item) {
        this.item = item;
    }
```

Vurder disse tre klassene. Hvilket konsept bør brukes for å forbedre denne koden?

Velg ett alternativ:

- Polymorfisme
- Arv
- Komposisjon
- Generiske typer
- Enkapsulering

Maks poeng: 2

2 Tetromino 1

Nedenfor har vi deler av Tetromino-koden fra semesteroppgave 1.

```
public class Tetromino implements Iterable<GridCell<Character>> {

    public char symbol;
    public boolean[][] shape;
    public CellPosition pos;

    private Tetromino(char symbol, boolean[][] shape, CellPosition pos) {
        this.symbol = symbol;
        this.shape = shape;
        this.pos = pos;
    }
```

Hvilket konsept bør brukes for å forbedre denne koden?

Velg ett alternativ:

- Polymorfisme
- Arv
- Komposisjon
- Generiske Typer
- Enkapsulering

Maks poeng: 2

3 Tetromino 2

Nedenfor har vi deler av Tetromino- og CellPosition-klassen fra semesteroppgave 1.
Tetromino:

CellPosition:

```
public class CellPosition {

    private int x;
    private int y;

    public CellPosition(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
}
```

Hvilket konsept bør brukes for å forbedre denne koden?
Velg ett alternativ:

- Polymorfisme
- Arv
- Komposisjon
- Generiske Typer
- Enkapsulering

Maks poeng: 2

4 Klasse VS Objekt

Pokemon:

Main:

Hva er forskjellen på en klasse, et objekt og en referanse? Bruk koden ovenfor som et eksempel.
Besvarelsen din bør være utdypet nok til at sensor kan avgjøre om du har forstått konseptene.
Skriv ditt svar her

forskjellen mellom en klasse og et objekt kan beskrives med en metafor: La oss si at du eier en pc fabrikk som lager pcer, da blir klassen oppskriften på en pc, denne oppskriften kan enten være veldig nøyaktig eller litt mer løs. Oppskriften kan nå bli brukt til å lage mange forskjellige pcer som kan ha litt ulike egenskaper(CPU modell, antall GB minne, osv) men alle skal grovt sett ha samme funksjonalitet.

Objektene blir unike med egne individuelle egenskaper og minneadresser mens klassen beskriver hva som er likhetene og den gjennerelle rammen til objektene (varaibler, metoder, osv). Et objekt kan også ha de nøyaktig samme egenskapene til et annet objekt men de er fortsatt to forskjellige objekter med to forskjellige minneadresser. Vi kaller også et objekt en instans av en klasse.

Hvert objekt kan tilhøre bare en klasse, og blir en samling av data hvor klassen bestemmer hvilke type data objektene skal bestå av.

For å forstå hvordan referanser fungerer og hva som det har med klasser og objekter og gjøre må vi først forstå hvordan minnet på en datamaskin er bygget opp. Minne blir hovedsaklig delt opp i to deler: stack og heap. Når et objekt blir laget (eks Computer lenovoThinkpad = new Computer()) så blir det reservert en plass på heapet for all dataen til dette lenovoThinkpad objektet.

På stacken lagres minneadressen til objektet og verdien den har. Det som er kult med denne verdien er at den ikke er selve daten til objektet med den refererer minnneverdien til hvor all daten ligger på heapen. Så Computer lenovoThinkpad referer egentlig bare til hvor på heapet dataen ligger.

Referanser blir brukt hos de datatypene vi kaller for referte datatyper, vi har også en annen datatype som heter primitive datatyper (int,char,boolean, osv.) her i motseting til refererte datatyper så er verdien til minneadressen den faktiske verdien.

Så for å oppsummere:
- klasse er en oppskrift som inneholder hvilke egenskaper og hvilke typer data objektene til klassen skal ha og man kan lage unike objekter (instanser) med hjelp av denne oppskriften.
- et objekt er individuelle instanser av en klasse og har egenskaper og egen data, lagret i instansvaraiblene, som alle følger oppskriften til klassen.
- en referanse kan tenkes på som pilen som peker fra stacken, i dette tilfellet fra minneadressen til objektet, ned på heapet hvor den faktiske dataen er lagret.

Ord: 392

Maks poeng: 10

5 Calculator Polymorfisme

Polymorfisme er et viktig konsept innen objektorientert programmering.

I kodesnutten nedenfor vises bruk av en kalkulator-klasse. I dette eksempelet regner vi ut eksponenten "10^3" (ti opphøyd i tredje), som resulterer i verdien 1000.

I lenken nedenfor har vi all koden for denne kalkulatoren. I tillegg til eksponent inneholder koden operasjonene addisjon (pluss), subtraksjon (minus) og multiplikasjon (ganging).

Lenke til kode

Se over koden og forklar hvordan polymorfisme er brukt.
Besvarelsen din burde være utdypet nok til at sensor kan avgjøre om du har forstått konseptene.

Skriv ditt svar her

polymorfisme betry direkte oversatt: "mange form". Dette gjør at vi kan ha en datatype som kan ta form som forskjellige klasser uten å endre på typen. Det første vi kan se etter når vi leter etter polymorfisme er om det er brukt en abstrakt klasse, eller et interface. Vi ser at her er det brukt et interface som heter: IOperation, som gir oss to metodesignaturer som en hver klasse som implementerer dette interfacet må ha: getSymbol() og calculate(). Klassene addition, subtraction, multiplication og exponent implementerer dette interfacet og lager egne "versjoner" av hva metodene i interfacet skal gjøre/returnere.

I calculator så brukes det objekter med typen IOperation, det vil si at alle objekter av klasser som implementerer IOpeation kan bli brukt her, og deres versjon av metodene vil bli kalt når metoden kalles.

På toppen av Calculator lages det et oppslagsverk som tar inn en string som key og et objekt objekt som implementerer IOperation, så her blir det brukt polymorfisme. Lenger nede ser vi denne polymorfismen igjen i addOperations(), hvor vi ser at vi kan legge til flere forskjellige klasser i et og samme oppslagsverk selv om java er statisk skrevet.

I den eneste public funksjonen calculate så ser vi hvordan polymorfismen blir brukt til å regne ut. Vi henter ut det objektet av den klassen som symbolet er key til ("+","-","*" eller "^") og utifra dette kan vi hente ut det objektet fra den klassen som vi trenger til å regne ut. Dette blir gjort på to linjer takket være polymorfisme som gjør at vi kan ha flere klasser som har fellesmetoder som gjør at vi kan bruke dem som en type. Merk at du kun kan bruke de metodene som er angitt i interfacet når du bruker polimorfisme.

Ord: 291

Maks poeng: 10

6 Calculator Tests

Vi har skrevet noen tester for Calculator koden i forrige oppgave.
Testene har som formål å teste om:
1) rett operator brukes på rett tegn
2) at begge variablene gis til operatoren i rett rekkefølge

Se på koden under og anta at vi har implementert en kalkulator som passerer alle testene.

F.eks. 0+0 er en dårlig test for å sjekke om rett operator er brukt siden 0+0 = 0*0 = 0-0 = 0^0.

Hvilke av følgende påstander er sann? Du kan velge flere påstander.

- [ ] canAdd() kan passere selv om koden implementerer feil operator
- [x] canMultiply() kan passere selv om koden implementerer feil operator
- [ ] canSubtract() kan passere selv om koden implementerer feil operator
- [x] canSubtract() kan passere selv om argumentene er i ugyldig rekkefølge
- [ ] canExponentiate() kan passere selv om koden implementerer feil operator
- [x] canExponentiate() kan passere selv om argumentene er i ugyldig rekkefølge

Maks poeng: 3

7 DNASequence

I denne oppgaven skal du implementere en klasse for å håndtere DNA sekvenser.
En DNA sekvens er en liste som kun inneholder bokstavene 'A','T','C','G', for eksempel: A,C,T,G,T,C,T,C,A,T

Vi skal lage en forenklet versjon av DNASequence så du trenger ingen kunnskaper om hva DNA er for å løse denne oppgaven.
Du får utdelt kode med noen TODO: i koden der du må legge til kode

Hint: både klassen String og klassen Character har metodene toUpperCase() og toLowerCase() som kan være nyttige i denne oppgaven.

Klikk på lenken under for å se koden som er utdelt, husk at du må kopiere koden inn i Inspera når du er ferdig.

DNASequence code

Det er 4 ting du må gjøre:

1. (5 poeng) Legg inn det som mangler i DNASequence.java for at MainDNA.java skal kjøre. Du skal ikke endre MainDNA.java, kun DNASequence.java
2. (5 poeng) Implementer metoden isValid()
3. (5 poeng) Implementer metoden countOccurences()
4. (5 poeng) Det er noen linjer i MainDNA som er kommentert ut (linje 35 og 36 samt metoden makeString()) kommenter inn disse.
   Nå kompilerer ikke koden, du må legge til nødvendig kode for å få MainDNA.java til å kompilere og kjøre igjen.

Du får poeng for at koden fungerer og at den følger de prinsippene vi har lært i kurset.
Rett tankegang er viktigere enn rett syntax.
Skriv ditt svar her

```
1  import java.util.Arrays;
2  import java.util.List;
3  import java.util.LinkedList;
4  import java.util.Iterator;
5
6  /**
7   * DNASequence class is used for storing and doing operations
8   * on a dna sequence.
9   * key operations:
10  * isValid()
11  * countOccurences()
12  */
13 public class DNASequence implements Iterable<Character>{
14     private String sequence;
15
16     /**
17      * The constructor of the DNASequence class.
18      * Saves the param String sequence as an instance vaiable.
19      * @param sequence representing the DNA.
20     */
21     public DNASequence(String sequence){
22         this.sequence = sequence.toUpperCase();
```

```
    }

    @Override
    public Iterator<Character> iterator(){
        List<Character> it = new LinkedList<>();
        for(int i = 0; i < this.sequence.length(); i++){
            it.add(sequence.charAt(i));
        }
        return it.iterator();
    }

    public static final List<Character> CODES = Arrays.asList('A','T','C','G');

    /**
     * This method checks is a String contains a valid DNA sequence.
     * Only letters in the list CODES can be in the string.
     * The method should ignore letter case e.g. "cat" and "CAT" and "Cat" should all be valid sequences
     * An empty string is not a valid string
     * @param sequence the string to check
     * @return true if the string only contains valid codes, false otherwise
     */
    public static boolean isValid(String sequence) {
        //TODO:
        if(sequence.length() == 0){
            return false;
        }
        String upperS = sequence.toUpperCase();
        for (int i = 0; i < upperS.length();i++){
           Character c = upperS.charAt(i);
           if(!CODES.contains(c)){
               return false;
           }
        }
        return true;
    }

    /**
     * This method counts the number of times a given code is present in this DNASequence object.
     * If the given char is not a valid code, the method will throw an IllegalArgumentException
     * The method should accept both upper case and lower case letters as input and ignore the case when counting
     * @param code
     * @return the number of times code occurs in this DNASequence
     */
    public int countOccurences(char code) {
        //TODO:
        Character codeChar = null;
        codeChar = codeChar.toUpperCase(code);
        if(!CODES.contains(codeChar)){
            throw new IllegalArgumentException("Invalid code");
        }
        int count = 0;

        for(int i =0; i < this.sequence.length(); i++){
            if(this.sequence.charAt(i) == (char) codeChar){
                count++;
            }
        }
        return count;
    }
}
```

Maks poeng: 20

8 SudokuBoard Arv vs Composition

For hver av de 4 kodene avgjør om de bruker arv eller Composition

Finn de som passer sammen:

	bruker arv	bruker Composition	ingen av delene	begge deler
SudokuBoardA	◉	○	○	○
SudokuBoardB	○	◉	○	○
SudokuBoardC	◉	○	○	○
SudokuBoardD	○	◉	○	○

Maks poeng: 4

9 SudokuBoard sizes

Vanlig sudoku spilles på 9x9 brett, men kan spilles på andre størrelser.

Hvilke av kodene kan håndtere brett av forskjellig størrelse uten å endre på koden?
Her ser vi kun på om størrelsen på brettene er rett, ikke innholdet.

Du får 1 poeng for rett valg, 0 poeng for blankt svar og -1 poeng for hvert feil valg.

Avgjør hva som er rett for hver av de 4 klassene.

	Håndterer alle størrelser	Håndterer kun noen størrelser
SudokuBoardA	◉	○
SudokuBoardB	◉	○
SudokuBoardC	○	◉
SudokuBoardD	◉	○

Maks poeng: 4

10 SudokuBoard Encapsulation

Innkapsling (encapsulation) er viktig for å unngå feil i senere utvidelse av koden.
Se på beskrivelsen av SudokuBoard på infoside tidligere i eksamen før du løser denne oppgaven.

Se over de 4 kodene og avgjør hvilke av klassene som har god innkapsling.

Innkapsling gjøres på flere nivåer av flere forskjellige årsaker.
I denne oppgaven fokuserer vi på om ugyldig tilstander kan oppstå i SudokuBoard ved at noen bruker klassen på feil måte.

Det vi mener med ugyldig tilstand i denne oppgaven er f.eks. hvis et 9x9 SudokuBoard kan inneholde verdien 11 eller andre ugyldige verdier kan finne veien inn i klassen.

Du får 1 poeng for rett valg, 0 poeng for blankt svar og -1 poeng for hvert feil valg.

For hver kode, avgjør om innkapsling er god eller hvilke feil som kan oppstå.

	God innkapsling	Ugyldig tilstand kan oppstå
SudokuBoardA	◉	○
SudokuBoardB	◉	○
SudokuBoardC	◉	○
SudokuBoardD	○	◉

Maks poeng: 4

11 SudokuBoard and IGrid

Alle 4 studentene har bygget på IGrid.java
Husk at vi har lært at en klasse som implementerer et Interface må implementere alle metodene i interfacet slik de oppfyller kravene som står i kommentarene til interfacet.

Hvilke av kodene oppfyller kravene i interfacet og hvem bryter med interfacet?

Hint: se nøye på kommentaren til get() og set() metodene i IGrid så du skjønner hva de skal gjøre og sjekk hver av de 4 kodene.

Du får 1 poeng for rett valg, 0 poeng for blankt svar og -0.5 poeng for hvert feil svar.

Avgjør hva som er rett for hver av de 4 klassene.

	Følger kravene til IGrid	Bryter med kravene til IGrid	Implementerer ikke IGrid
SudokuBoardA	○	◉	○
SudokuBoardB	◉	○	○
SudokuBoardC	○	○	◉
SudokuBoardD	◉	○	○

Maks poeng: 4

12 Poeng fra Semesteroppgavene V24

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

Du er nå ferdig med eksamen :-)

God Sommer!

hilsen Martin

Skriv en hyggelig melding til foreleser ;-)

bra undervist:)

Maks poeng: 35