



KANDIDAT

10107

PRØVE

TDT4100 1 Objektorientert programmering

Emnekode	TDT4100
Vurderingsform	Hjemmeeksamen
Starttid	15.04.2021 10:00
Sluttid	15.04.2021 12:00
Sensurfrist	--
PDF opprettet	16.04.2021 15:03

Introduksjon

Oppgave	Oppgavetype
---------	-------------

i	Dokument
----------	----------

Del 1

Oppgave	Oppgavetype
---------	-------------

1	Flervalg
---	----------

2	Flervalg
---	----------

3	Flervalg
---	----------

4	Flervalg
---	----------

5	Flervalg (flere svar)
---	-----------------------

6	Flervalg
---	----------

Del 2

Oppgave	Oppgavetype
---------	-------------

7	Flervalg (flere svar)
---	-----------------------

8	Flervalg
---	----------

9	Flervalg
---	----------

10	Flervalg (flere svar)
----	-----------------------

11	Flervalg (flere svar)
----	-----------------------

12	Flervalg
----	----------

13	Flervalg
----	----------

14	Flervalg
----	----------

15

Flervalg

Del 3**Oppgave****Oppgavetype**

16

Flervalg (flere svar)

17

Flervalg (flere svar)

18

Flervalg

19

Flervalg

20

Flervalg

Del 4**Oppgave****Oppgavetype**

21

Fyll inn tekst

22

Paring

1 NB: Oppgaven inneholder to spørsmål

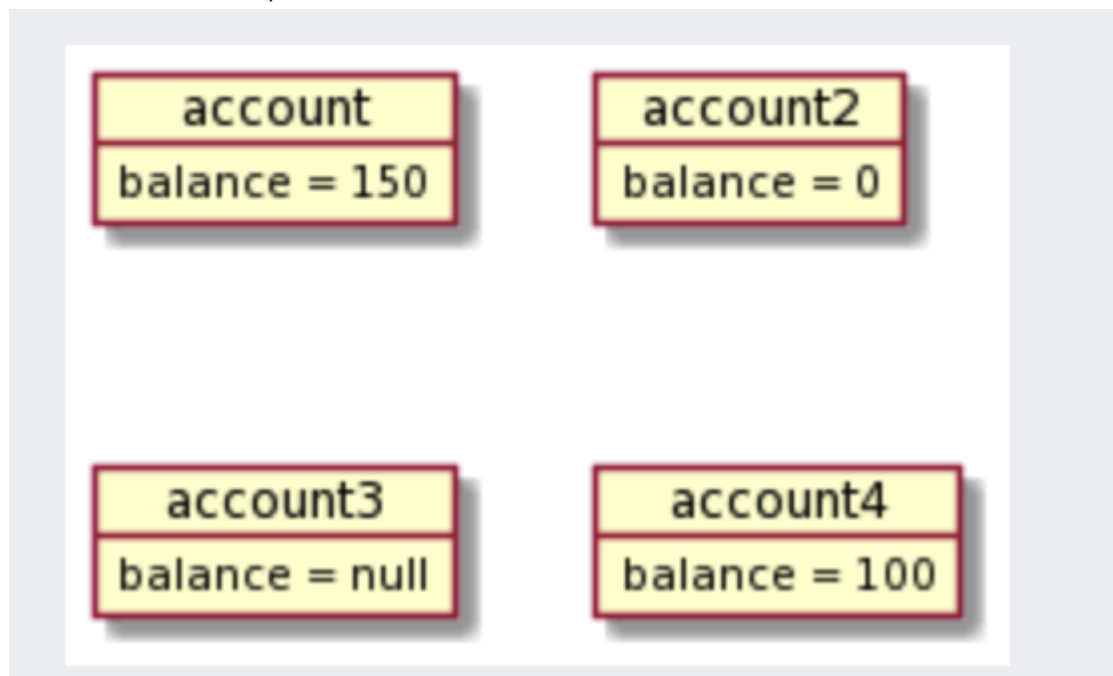
Ta utgangspunkt i følgende klasse:

```
public class Account {  
    private int balance;  
  
    public void withdraw(int amount) {  
        this.balance -= amount;  
    }  
  
    public void deposit(int amount) {  
        this.balance += amount;  
    }  
}
```


Etter kjøring av følgende kode:

```
Account account = new Account();
```

Hvilket diagram stemmer med objekttilstanden? (Du trenger ikke tenke på navnet på objektet, kun på tilstanden i **balance**)



Velg ett alternativ:

- ☐ account
- ☒ account2 
- ☐ account3
- ☐ account4


Deretter kjører du følgende kodelinjer:

account.deposit(200);

account.withdraw(50);

Hvilket diagram stemmer nå med tilstanden?

Velg ett alternativ

- ☒ account 
- ☐ account2
- ☐ account3
- ☐ account4

2 For en klasse definert som:

```
public final class DoesSomething {  
    ...  
}
```

Hvilken synlighetsmodifikator vil det i de fleste tilfeller være hensiktsmessig å bruke på feltene i klassen?

Velg ett alternativ:


- ☒ **private** ✓
- ☐ **public**
- ☐ **protected**
- ☐ Ingen modifikator

3 Alle klasser unntatt _____ arver fra en klasse




Velg ett alternativ:

- ☐ String
- ☒ **Object** ✓
- ☐ Character
- ☐ Integer

4 Når bør du bruke **final** i deklarasjonen til et felt?**Velg ett alternativ:**

- ☒ Når feltet aldri skal endre verdi 
- ☐ Når feltet er av typen **List**
- ☐ Når feltet skal kunne være **null**
- ☐ Når feltet ikke skal ha en *get-metode*

5 I hvilke(n) sammenheng(er) må du bruke **Integer** istedenfor **int**?**Velg ett eller flere alternativer**

- ☐ Hvis verdien skal ha desimaltall
- ☒ Når du deklarerer en **List** 
- ☒ Hvis verdien skal kunne være **null** 
- ☒ Hvis verdien skal kunne konverteres fra en **String** 
- ☐ Hvis verdien skal kunne være negativ

- 6 Gitt en klasse **House**, som skal ha oversikt over en samling objekter av klassen **Room** i feltet **rooms**. Hva er det mest hensiktsmessig å deklareere feltet **rooms** som gitt at dette er alt vi vet om klassen?

Velg ett alternativ:

- ☐ List rooms = new List<>();
- ☐ List rooms = new ArrayList<>();
- ☐ ArrayList rooms = new ArrayList<>();
- ☒ Collection rooms = new ArrayList<>(); ✓

- 7 Følgende fire felt har disse fire **set**-metodene. Hvilke(n) av **set**-metodene er et brudd på god innkapslingsskikk?

```
private String text;
private int number;
private char character;
private List<String> list = new ArrayList<>();

public void setText(String text) {
    this.text = text;
}
public void setNumber(int number) {
    if (number > 0) {
        this.number = number;
    }
}
public void setCharacter(char character) {
    this.character = character;
}
public void setList(List<String> list) {
    this.list = list;
}
```


Velg ett eller flere alternativer

- ☒ setList ✓
- ☐ setNumber ✓
- ☒ setText ✗
- ☐ setCharacter

- 8 Du har to klasser, **Student** og **ExchangeStudent**. Disse deler mange implementasjons-detalljer, men varierer noe på andre. Blant annet er hver **Student** tilknyttet et visst institutt og fakultet, mens **ExchangeStudent** ikke er det. Du ønsker å kunne dele mest mulig kode mellom disse klassene i implementasjonen din.


Hvilken teknikk er best egnet til å løse dette problemet?

Velg ett alternativ:

- ☐ Observatør-observert
- ☐ Innkapsling
- ☐ Grensesnitt
- ☒ Arv 
- ☐ Delegering

- 9 En metode skal ved hjelp av en **Comparator** sortere en liste av klassen **Student** basert på kandidatnummer fra minst til størst. Hva bør **Comparator-metoden** returnere når den sammenligner 001 (kandidatnummer til første objekt) mot 110 (kandidatnummer til andre objekt) ?

Velg ett alternativ:

- ☒ -1 
- ☐ Et hvilket som helst positivt tall
- ☐ 1
- ☐ 0

10 Gitt følgende kode:

```
public class A {  
    private B b;  
    public B getB() {  
        return b;  
    }  
}  
  
public void doSomething(List<A> list, Consumer<B> consumer) {  
    //implementation...  
}  
}
```

Hvilke(n) kodelinje(r) som erstatter (//implementation) vil tilfredstille typen til **consumer** og vil kompilere?

Velg ett eller flere alternativer

- ☐ list.stream().forEach(consumer);
- ☒ list.stream().map(A::getB).forEach(consumer); ✓
- ☐ list.stream().map(a => a.getB()).peek(consumer) ✓
- ☐ liste.stream().peek(consumer);

11 Hva er gyldig å skrive på høyresiden av uttrykket **y instance of x**. (Hva er gyldige substitutter for **x**).

Velg ett eller flere alternativer

☐ En primitiv type

☐ Et objekt

☐ En **boolean**

☒ Et **interface**



☒ En klasse



12 NB: Oppgaven inneholder to spørsmål

For en assosiasjon mellom klassen **Student** som har en tilstand med navn, og en klasse **ExamResult** med tilstand karakter. Hva vil best beskrive assosiasjonen?

Velg ett alternativ:

- ☐ mange-mange
- ☒ en-mange/mange-en
- ☐ en-en



For en assosiasjon mellom klassen **Country**, og klassen **President** som representerer presidenten i landet. Hva vil best beskrive assosiasjonen sett fra **President**?

Velg ett alternativ



- ☐ mange-mange
- ☐ en-mange/mange-en
- ☒ en-en



13 Hva er tilstanden til **number** i **test**-objektet etter at følgende kode er kjørt:

```
public class SimpleConstructor {  
    private int number;  
  
    public SimpleConstructor(int number) {  
        this.validateNumber();  
        this.number = number;  
    }  
  
    public void validateNumber() {  
        if (number < 0) {  
            throw new IllegalArgumentException("Can't have negative numbers");  
        }  
    }  
  
    public static void main(String[] args) {  
        SimpleConstructor test = new SimpleConstructor(-5);  
    }  
}
```

Velg ett alternativ:

- ☒ Det vil utløses et unntak 
- ☐ Koden vil ikke kompilere
- ☐ 0
- ☐ -5 

- 14 Du har to klasser, **Student** og **Course**. Hver gang feltet **examDate** til et **Course**-objekt endrer seg ønsker du å gi beskjed til alle studentene som er meldt opp til eksamen i emnet om endringen.

Hvilken teknikk er best egnet til å løse dette problemet?

Velg ett alternativ:

- ☐ Grensesnitt
- ☐ Innkapsling
- ☐ Arv
- ☒ Observatør-observert ✓
- ☐ Delegering

- 15 Basert kun på følgende kode, hva kan vi vite er feil med denne metoden?

```
public boolean validateNumberIsCorrectFormat(String number) {  
    if (!NumberValidator.isNumberCorrectFormat(number)) {  
        throw new IllegalArgumentException("Wrong format");  
    }  
    if (NumberValidator.isInvalidValue(number)) {  
        throw new IllegalArgumentException("Illegal value");  
    }  
}
```

Velg ett alternativ:

- ☐ **NumberValidator** er ikke opprettet og vi kan derfor ikke bruke metodene derfra
- ☒ Metoden returnerer ikke noe ✓
- ☐ Metoden setter ikke **number** til å være noen verdi
- ☐ Metoden burde utløse **IllegalStateException**

16

Anta følgende klasser, grensesnitt og metoder:

- Klasse **K** deklarerer metoden **K methodK()**
- Grensesnittet **G** deklarerer metoden **G methodG(K)**
- Klasse **M** arver fra **K**, implementerer **G** og deklarerer metoden **M methodM(K)**
- Klasse **C** implementerer **G** og deklarerer metoden **C methodC(G)**

Hvilke(n) av følgende deklarasjoner vil **ikke** compilere:

Velg ett eller flere alternativer

☒ G g1 = new K(); ✓

☐ G g3 = new C();

☐ M m = new K(); ✓

☐ K k = new M();

☒ G g2 = new M(); ✗

- 17** Hvilke(n) av disse metodene kan være hensiktsmessig å ha med for den observerte parten i observatør-observert-teknikken?

Velg ett eller flere alternativer

☒ fireStateChanged ✓

☐ updateState

☐ listen

☒ addListener ✓

☒ removeListener ✓

18 Hvilken metode tilfredsstiller følgende valideringsregler:

1. Tallet skal ikke være **null** eller negativt
2. Hvis det er negativt eller **null** skal **IllegalArgumentException** utløses
3. Tallet kan ikke være mer enn 10. Det skal da returneres **false**
4. Det skal ellers returnere **true**. (Tallet er fra og med 0 til og med 10)

```
public boolean validateNumber1(int number) {  
    if (number < 0) {  
        throw new IllegalArgumentException("Not valid");  
    }  
    return number <= 10;  
}  
  
public boolean validateNumber2(Integer number) {  
    if (number == null || number < 0) {  
        throw new IllegalArgumentException("Not valid");  
    }  
    return number <= 10;  
}  
  
public boolean validateNumber3(Integer number) {  
    return number != null && number <= 10 && number > 0;  
}  
  
public boolean validateNumber4(int number) {  
    if (number < 0 || number == null) {  
        throw new IllegalArgumentException("Not valid");  
    }  
    if (number >= 10) {  
        return false;  
    }  
    return true;  
}
```

Velg ett alternativ:

- ☐ validateNumber1
- ☒ validateNumber2 ✓
- ☐ validateNumber3
- ☐ validateNumber4

19 NB: Oppgaven inneholder to spørsmål

I følgende implementasjon av en selv-refererende 1:1-assosiasjon kan det ha sniket seg inn en feil (bug):

```
public class SelfReference {  
  
    private SelfReference relation;  
  
    public SelfReference getRelation() {  
        return this.relation;  
    }  
  
    public void setRelation(SelfReference relation) {  
        if (relation == this.relation) {  
            return;  
        }  
        this.relation = relation;  
        SelfReference oldRelation = this.relation;  
        relation.relation = this;  
        oldRelation.relation = null;  
    }  
}
```

Hva stemmer om tilstand i objektene etter kjøring av følgende kodelinjer:

```
SelfReference example1= new SelfReference();  
SelfReference example2 = new SelfReference();  
SelfReference example3 = new SelfReference();  
example1.setRelation(example2);  
example1.setRelation(example3);
```

Velg ett alternativ:

- ☐ Alt vil ha korrekt tilstand i dette eksemplet
- ☐ example1 har feil tilstand
- ☐ example2 har feil tilstand
- ☒ example3 har feil tilstand 
- ☐ Alle objektene vil ha feil tilstand
- ☐ example1 og example3 vil ha feil tilstand
- ☐ example1 og example2 vil ha feil tilstand
- ☐ example2 og example3 vil ha feil tilstand
- ☐ Det vil utløses ein **NullPointerException**

Dersom du hadde byttet om på de to siste linjene i **setRelation**, slik at det ble:

```
oldRelation.relation = null;  
relation.relation = this;
```

Hva hadde nå stemt om tilstanden til objektene etter kjøring av de samme kodelinjene?



Velg ett alternativ

- ☐ Alt vil ha korrekt tilstand for dette eksempelet
- ☐ example1 har feil tilstand
- ☒ example2 har feil tilstand 
- ☐ example3 har feil tilstand
- ☐ Alle objektene vil ha feil tilstand
- ☐ example1 og example3 vil ha feil tilstand
- ☐ example1 og example2 vil ha feil tilstand
- ☐ example2 og example3 vil ha feil tilstand
- ☐ Det vil utløses en **NullPointerException**

20

Gitt en JavaFX-kontroller UIController, en FXML-fil UI.fxml og en modell Model. Brukergrensesnittet skal oppdatere seg hver gang tilstanden i et Model-objekt endrer seg. Du tenker å løse dette ved hjelp av observatør-observert teknikken. Hvem er den observerte i denne sammenhengen?

Velg ett alternativ:

- ☐ UI.fxml
- ☐ UIController
- ☒ Model 
- ☒ Det gir ingen mening å bruke observatør-observert 

- 21** Fyll inn det som mangler for at **1:1**-assosiasjonen mellom to **Person**-objekter skal fungere. Svaret skal kun bestå av ett kodeord/en variabel, ingen mellomrom eller mer kompliserte uttrykk.

```
public class Person {  
  
    private Person supervisor;  
    private Person employee;  
  
    public void setSupervisor(Person supervisor) {  
        if (superVisor == this.supervisor) {  
             ✓  
        }  
        Person oldSupervisor = this.supervisor;  
        this.supervisor =  ✓  
        if (oldSupervisor != null && oldSupervisor.getEmployee() == this) {  
            oldSupervisor.setEmployee(null);  
        }  
        if (this.supervisor != ) { ✓  
            this.supervisor.setEmployee(this);  
        }  
    }  
}
```

22 Gitt klassen **Partner** med følgende implementasjon

```
public class Partner {  
    private Partner partner;  
  
    public Partner getPartner() {  
        return partner;  
    }  
  
    public void setPartner(Partner partner) {  
        if (partner == this.partner) {  
            return;  
        }  
        Partner oldPartner = this.partner;  
        this.partner = partner;  
        if (oldPartner != null && oldPartner.getPartner() == this) {  
            oldPartner.setPartner(null);  
        }  
        if (this.partner != null) {  
            this.partner.setPartner(this);  
        }  
    }  
}
```

Følgende kode blir kjørt:

```
Ola ola = new Partner();  
Kari kari = new Partner();  
Petter petter = new Partner();  
Marit marit = new Partner();
```

```
ola.setPartner(kari);  
kari.setPartner(petter);  
marit.setPartner(ola);  
kari.setPartner(null);
```

For hvert av objektene over, hva peker **this.partner** til dette objektet på?

Eksempel: Hvis du mener at **ola.partner** peker på **petter**, huker du av **petter** i første rad. Hvis du mener **ola.partner** er **null**, huker du av **null**.

	marit	ola	null	kari	petter
kari	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/> ✓	<input type="radio"/>	<input type="radio"/>
ola	<input checked="" type="radio"/> ✓	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
petter	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/> ✓	<input type="radio"/>	<input type="radio"/>
marit	<input type="radio"/>	<input checked="" type="radio"/> ✓	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>