Høgskolen i Østfold

EKSAMEN

Emnekode:	**Emnenavn:**
ITF10611	Objektorientert programmering
Dato: 3. mai 2018	**Eksamenstid:** 4 timer
Hjelpemidler: To A4-ark (fire sider) med egne notater	**Faglærer:** Lars Emil Knudsen

Om eksamensoppgaven og poengberegning:

Oppgavesettet består av 15 sider inklusiv denne forsiden og vedlegg. Du er selv ansvarlig for å kontrollere at oppgaven er komplett før du begynner å besvare spørsmålene. Det er på hver hovedoppgave angitt hvor mye disse teller av totalen. Karakter fastsettes på grunnlag av en helhetsvurdering av besvarelsen.

I oppgavene hvor du blir bedt om å skrive kode anbefales det at du skriver løsningen med java- syntaks. Er du derimot usikker på hvordan du skal besvare en oppgave med kode kan du skrive svaret med egne ord, det er da viktig at du beskriver logikken på en omfattende og detaljert måte.

Om en oppgave virker å forutsette at du har løst en tidligere oppgave så kan du forutsette at denne er løst uavhengig om du faktisk har gjort det eller ei. F.eks: om du i en oppgave må benytte metoden: metode() som du skulle ha implementert i en tidligere oppgave, så kan du bare anta at dette er gjort og den fungerer slik det er tenkt.

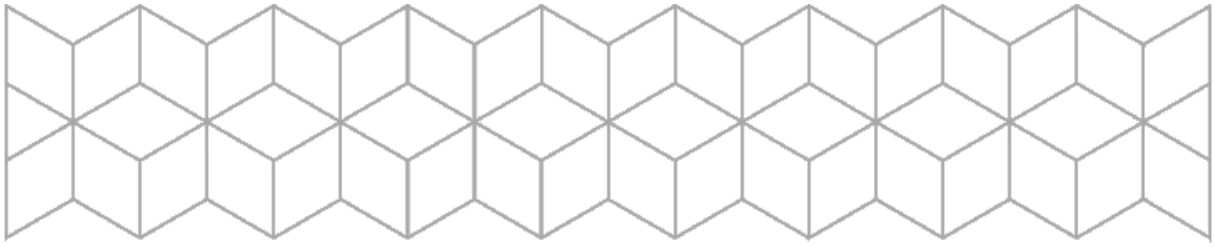
Pass også på å svare på alle oppgaver. Det er bedre å skrive litt i grove trekk hvordan du ser for deg at oppgaven kan løses, enn å ikke skrive noe i det hele tatt, dersom du står fast.

Takk for et veldig hyggelig semester. Lykke til med videre eksamener og studieløp. :)

Lykke til!

Sensurfrist: 24. mai 2018

Karakterene er tilgjengelige for studenter på Studentweb www.hiof.no/studentweb



Oppgave 1 – 20%

Disse oppgavene skal besvares kort og presist. Du trenger altså ikke skrive en liten stil på hver av dem, men pass på at du besvarer alt oppgaven spør etter. Det vil bli lagt vekt på at forklaringen er skrevet med dine ord, og at den ikke er avskrift fra andre kilder. Det er fordelaktig å lage eksempler med kode og/eller illustrasjoner.

Oppgave 1.1 – Interface

Forklar hva et interface er, og hva forskjellene/likhetene på et interface og en abstrakt klasse er. Eksemplifiser med enkel kode.

Oppgave 1.2 – Primitive Datatyper

Forklar hva primitive datatyper er, nevn gjerne noen eksempler.

Oppgave 1.3 - Polymorfisme

Forklar begrepet polymorfisme og eksemplifiser med enkel kode.

Oppgave 1.4 - Innkapsling

Forklar hva innkapsling er og hvorfor prinsippet benyttes i OOP. Eksemplifiser med enkel kode.

OPPGAVE 2 – 30%

Del 2 tar hovedsakelig for seg analyse og forståelse av Javakode og dens oppbygning.

OPPGAVE 2.1

Oppgaven inneholder en eller flere feil. Hvilke av disse linjene merket A-E er korrekt/ikke korrekt? Forklar eventuelt hvorfor og skill på om det er en kompileringsfeil, eller om det først vil feile når programmet kjøres.

```
class Dyr {   }
class Pattedyr extends Dyr {   }
class Primat extends Pattedyr {   }

public static void main(String[] args) {
        Dyr dyr1;
        Pattedyr pattedyr1, pattedyr2;
        Primat primat1, primat2;

        pattedyr1 = new Pattedyr();
        primat1 = new Primat();

        dyr1 = primat1;                 // A
        primat2 = (Primat) pattedyr1;    // B
        pattedyr2 = (Dyr) pattedyr1;     // C
        pattedyr2 = (Pattedyr) primat1;  // D
        primat1 = pattedyr2;            // E
}
```

Oppgave 2.2

I disse deloppgavene skal du skrive hvilken utskrift koden i mainmetoden resulterer i.

Oppgave 2.2.1

```
class Dyr
{
        private static int id = 0;
        private String navn;
        private int aarOppdaget;
        private int populasjon;

        public Dyr(String navn, int aarOppdaget, int populasjon)
        {
            id = id + 1;
            this.navn = navn;
            this.aarOppdaget = aarOppdaget;
            this.populasjon = populasjon;
        }

    public String toString()
    {
       return "Navn: " + navn + " - Id: " + id;
    }

}

public static void main(String[] args) {
        List<Dyr> dyreListe = new ArrayList<>();
        dyreListe.add(new Dyr("Europeisk Bison", 1758, 3200));
        dyreListe.add(new Dyr("Bongo", 1902, 28000));
        dyreListe.add(new Dyr("Fjellgaselle", 1766, 15000));

        for (Dyr etDyr : dyreListe)
            System.out.println(etDyr);
}
```

OPPGAVE 2.2.2

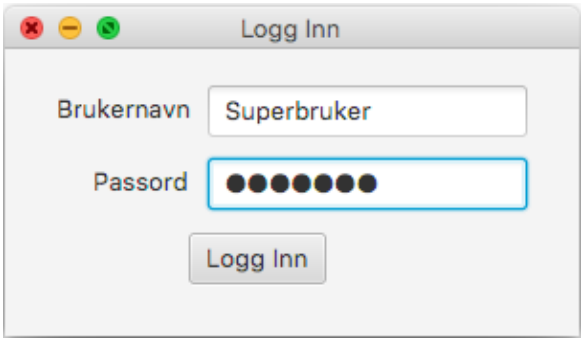
```
public static void main(String[] args) {
        // Vi bruker klassen Dyr som ble definert i forrige oppgave
        List<Dyr> dyreListe = new ArrayList<>();
        dyreListe.add(new Dyr("Europeisk Bison", 1758, 3200));
        dyreListe.add(new Dyr("Bongo", 1902, 28000));
        dyreListe.add(new Dyr("Nyala", 1758, 33000));
        dyreListe.add(new Dyr("Fjellgaselle", 1766, 15000));

        Collections.sort(dyreListe, new Comparator<Dyr>() {
            @Override
            public int compare(Dyr dyr1, Dyr dyr2) {
              int forskjell = dyr1.getAar() - dyr2.getAar();

              if (forskjell == 0) {
                return dyr2.getPopulasjon()-dyr1.getPopulasjon();
                }

              return forskjell;
            }
        });


        for (Dyr etDyr : dyreListe)
            System.out.println(etDyr.getNavn());
        }

}
```

OPPGAVE 2.2.3

```
class Ol {
    public String navn(String enString) {
            return "Øl: " + enString;
    }
}

class Ale extends Ol {
    public String navn(Object etObjekt) {
            return "Ale: " + etObjekt;
    }
}

class PaleAle extends Ale {
    public String navn(String enString) {
            return "PaleAle: " + enString;
    }
}

public static void main(String[] args) {
        ArrayList<Ol> olListe = new ArrayList<>();

        olListe.add(new Ol());
        olListe.add(new Ale());
        olListe.add(new PaleAle());

        for (Ol enOl : olListe)
            System.out.println(enOl.navn("Java"));
}
```

OPPGAVE 2.3

Du har et vindu for å logge inn med navn og passord definert i LogInn.fxml. Denne skal benyttes i en tilhørende LoginnController.java.



a) Skriv kode for å sette opp feltvariabler for de to tekstfeltene og knappen i LogInnController.java.
b) Skriv kode for å håndtere et trykk på knappen via en anonym klasse. Når man trykker på knappen skal brukernavn og passordet skrives ut til konsollen.
c) Nevn en annen måte du kan koble sammen et trykk på en knapp mellom .fxml og en tilhørende JavaFX Controller.

LOGGINN.FXML

```
<AnchorPane prefHeight="200.0" prefWidth="400.0"
            xmlns="http://javafx.com/javafx/9"
            xmlns:fx="http://javafx.com/fxml/1"
            fx:controller="no.hiof.eksamen.oppgave2.4.LoggInnController">
  <children>
    <Label text="Brukernavn" />
    <Label text="Passord" />

    <TextField fx:id="brukernavnTextField" />
    <PasswordField fx:id="passordField" />

    <Button fx:id="loggInnKnapp" text="Logg Inn" />
  </children>
</AnchorPane>
```

LOGGINNCONTROLLER.JAVA

```
public class LoggInnController {
   // 3.1 a

   @FXML
   public void initialize() {
      // 3.1 b
   }
}
```

OPPGAVE 3 – 50%

Moren til Ola (9 ½ år) og Nils (7) har sett seg lei på at de bruker for mye tid til spilling på nettbrettet og i tillegg lyver om det. Hun ønsker derfor å kunne overvåke, logge og begrense bruken av nettbrettet til spilling. Vi har startet utviklingen av en prototype kalt MomBridgeAnalytica.

Vi har generalisert den noe, slik at den kan utvides til å håndtere andre app typer enn spill i fremtiden, men slik det er nå skal den bare kunne:

- Holde orden på hvor mye tid det er brukt på spill per bruker per dag
- Begrense spillbruken
- Logge spillbruken

I noen tilfeller vil det være naturlig å vise en melding til brukeren, vi har ikke koblet dette sammen med et brukergrensesnitt, så i denne oppgaven vil det være tilstrekkelig å skrive ut meldinger til konsollen.

Klassene som er laget i systemet så langt er:

- Main
  - Et typisk scenario for Ola sin bruk av nettbrettet. Denne representerer en service som kjører kontinuerlig, som ikke er en del av denne prototypen, men vi kan tenke oss eksisterer. Den vil gi beskjed til SpillBegrenseren når en app har startet og avsluttet, samt ved jevne mellomrom hvor lenge appen har kjørt. Det forutsettes også at SpillBegrenseren blir resatt hver natt slik at den starter spilloversikten på nytt.
- Bruker
  - Representerer den innloggede brukeren i systemet (navn, alder).
- App
  - Representerer en applikasjon (navn og tid brukt i dag).
- Spill
  - Representerer et spill på nettbrettet (har i tillegg kategori og alder)
- AppOvervaaker
  - Interface med generelle metoder for å få beskjed om informasjon om kjøringen av en app (appStartet(), harBruktAppGittTid(), appAvsluttet())
- SpillBegrenser
  - Implementerer AppOvervaaker, holder orden på hvor lenge hvert spill er brukt i dag, samt totalbruk av spill i dag.
  - Når et *Spill* blir startet skal den legges til i listen over spill forutsatt at:
    - Brukeren er gammel nok til å spille spillet (hvis ikke skal det skrives ut en melding om at brukeren ikke er gammel nok og spillet avsluttes)

        - Den totale dagsbruken ikke overskredet (hvis den er det, skal en advarsel genereres og vises og spillet avsluttes)
    - Hver gang det blir rapportert om hvor lenge et spill har vært brukt (harBruktAppGittTid()), skal spillets tid for bruk i dag oppdateres.
        - Hvis den totale dagsbruken er overskredet, skal en advarsel genereres og vises, samt spillet avsluttes
    - Når et *Spill* avsluttes skal spillets tid for bruk i dag oppdateres
        - Hvis den totale dagsbruken er overskredet, skal en advarsel genereres og vises.
            - Hvis ikke skal en melding om gjenværende spilletid vises.
- SpillLogger
    - Klasse som skriver en dags spillbruk til fil. Videre håndteringen av denne loggen er ikke en del av denne prototypen.

Oppgave 3.1

Lag et klassediagram, det er ikke nødvendig å ta med felter og metoder – kun klassenavn, over alle klassene i systemet hvor du tydelig får frem eventuelle arverelasjoner, implementasjoner og assosiasjoner. Du trenger ikke tegne inn Main-klassen.

Oppgave 3.2

Fyll ut konstruktøren for klassene Bruker og Spill.

Oppgave 3.3

Vi skal kunne sortere appene basert på tiden som er benyttet i synkende rekkefølge (den som er benyttet lengst kommer øverst). Gjør endringer/tillegg i klassen App for å legge til den naturlige sorteringsordenen.

Oppgave 3.4

Vi skal kunne finne den totale tiden som er benyttet på spill hittil i dag. Implementer metoden **getTotalDagsbrukFremTilNaa()**.

Oppgave 3.5

Implementer metoden **genererAdvarselNaarTidErOppbrukt()**. Denne skal returnere en melding på følgende format (spillene skal listes ut i synkende rekkefølge basert på tid brukt):

Hei Ola!
Du har dessverre brukt opp all tiden din på 180 minutter i dag.
Larsio Bros brukt i 2 timer 30 minutter
AngryChickens brukt i 0 timer 53 minutter

OPPGAVE 3.6

Vi skal kunne håndtere overvåkningen i SpillBegrenser ved hjelp av implementasjonene av AppOvervaaker interfacet. Se beskrivelsen av SpillBegrenser for mer informasjon om hvordan disse skal håndteres.

a) Implementer logikken for **appStartet()**
b) Opprett og implementer logikken for **harBruktAppGittTid()**
c) Opprett og implementer logikken for **appAvsluttet()**

OPPGAVE 3.7

Vi ønsker å kunne skrive en rapport over dagens spillforbruk. Implementer metoden **skrivSpillTilFil()** i klassen SpillLogger. Denne skal ta en liste med spill som skal skrives til en .csv fil. Hver linje skal inneholde navnet på spillet, kategori, aldersgrense og tiden brukt i dag, adskilt med semicolon **;.**

VEDLEGG TIL OPPGAVE 3

Klasser/Interface

- Main – Trenger ikke være med i UML
- App
- AppOvervaaker
- Bruker
- Spill
- SpillBegrenser
- SpillLogger

Vedlegget til oppgave 3 er på 5 sider inkludert denne siden

Main

```
public class Main {

  public static void main(String[] args) {
    // Installerte spill og apper på enheten
    // (blir initialisert på nytt hver natt)
    Spill angryChickens = new Spill("AngryChickens", "Puzzle", 3, 0);
    Spill fortDay = new Spill("FortDay", "FPS", 16, 0);
    Spill larsioBros = new Spill("Larsio Bros", "Platform", 3, 0);

    // Innlogget bruker
    Bruker brukerOla = new Bruker("Ola", 11);

    // En ny instans av spillBegrenser blir laget når en bruker logger
    inn for første gang for den dagen
    SpillBegrenser spillBegrenser = new SpillBegrenser(brukerOla, 180);


    /* Vi har en service som gjør kallene under på riktige tidspunkt,
    dette er et eksempel på hvilke kall den vil gjøre */
    // En app blir startet
    spillBegrenser.appStartet(angryChickens);

    // SpillBegrenseren opplyst om at en appbruk hvert 10 min
    for (int i=1; i < 5; i++)
       spillBegrenser.harBruktAppGittTid(angryChickens, 10*i);

    // Appen blir avsluttet
    spillBegrenser.appAvsluttet(angryChickens, 53);


    spillBegrenser.appStartet(fortDay);

    for (int i=0; i < 5; i++)
       spillBegrenser.harBruktAppGittTid(fortDay, 10*i);

    spillBegrenser.appAvsluttet(fortDay, 57);


    spillBegrenser.appStartet(larsioBros);

    // SpillBegrenseren opplyst om at en appbruk hvert 10 min
    // Grensen på 180 minutter blir nådd og spill blir avsluttet
    for (int i=0; i < 19; i++)
       spillBegrenser.harBruktAppGittTid(larsioBros, 10*i);


    // Kjøres ved midnatt
    // All dagens spillbruk blir logget og all spillbruk resatt
    SpillLogger.skrivSpillTilFil(spillBegrenser.getSpillListe(),
                        LocalDate.now() + "_" + brukerOla.getNavn() + ".csv");

    spillBegrenser = new SpillBegrenser(brukerOla, 180);
  }
}
```

App

```
public abstract class App {
    private String navn;
    private int tidBruktIDag;

    public App(String navn, int tidBruktIDag) {
        this.navn = navn;
        this.tidBruktIDag = tidBruktIDag;
    }

    // Oppgave 3.3


    // get/set-metoder finnes
}
```

AppOvervaaker

```
public interface AppOvervaaker {
    void appStartet(App app);
    void harBruktAppGittTid(App app, int tidAppHarVaertAktiv);
    void appAvsluttet(App app, int tidAppHarVaertAktiv);
}
```

Bruker

```
public class Bruker {
    private String navn;
    private int alder;

    public Bruker(String navn, int alder) {
        // Oppgave 3.2
    }

    // get/set-metoder finnes
}
```

Spill

```
public class Spill extends App {
    private String kategori;
    private int aldersgrense;

    public Spill(String navn, String kategori, int aldersgrense,
                                                        int tidSpiltIDag) {
        // Oppgave 3.2
    }

    // get/set-metoder finnes
}
```

SpillBegrenser

```
public class SpillBegrenser implements AppOvervaaker {
    private List<Spill> spillListe;
    private int maxMinutterLovligPaaEnDag = 180;
    private Bruker bruker;

    public SpillBegrenser(Bruker bruker,int maxMinutterLovligPaaEnDag){
        spillListe = new ArrayList<>();
        this.bruker = bruker;
        this.maxMinutterLovligPaaEnDag = maxMinutterLovligPaaEnDag;
    }

    @Override
    public void appStartet(App app) {
      // Oppgave 3.6a
    }

    // Oppgave 3.6b

    // Oppgave 3.6c

    public int getTotalDagsbrukFremTilNaa() {
        // Oppgave 3.4
    }


    private String genererAdvarselNaarTidErOppbrukt() {
        // Oppgave 3.5
    }

    // get/set-metoder finnes
}
```

SpillLogger

```
public class SpillLogger {

    public static void skrivSpillTilFil(List<Spill> spillListe,
                                                                    String filnavn) {
        // Oppgave 3.7
    }

}
```