Oblig 2

Nytt forsøk

Forfall 3. feb. av 23.59 **Poeng** 1 **Må leveres** en filopplasting

Alle oppgavene skal være forsøkt løst etter beste evne (du *må* ikke løse bonusoppgavene hvis det er noen). Hvis du har en oppgave som du har forsøkt på, men ikke fått helt til, er det OK om denne kommenteres ut slik at vi kan se hvordan du har tenkt/hva du har forsøkt på. Resterende kode bør kunne kompileres og kjøres uten at det kræsjer. Oppgaver som er gjennomført og fungerer burde *ikke* kommenteres ut.

Les: Obligatoriske oppgaver - Informasjon (https://hiof.instructure.com/courses/7058/pages/obligatoriske-oppgaver-informasjon)

Obligen er strukturert opp i oppgaver, dere skal levere alle filene samlet.

Teorispørsmålene: Leveres som docx/pdf/txt navngitt: Oblig2_TV_<DittNavn>.<filtype>
Eks: Oblig2_TV_LarsEmilKnudsen.pdf

Programeringsoppgavene: Du skal levere **.java** filene som en .zip fil navngitt: Oblig2_TV_<DittNavn>.zip
Eks: Oblig2_TV_LarsEmilKnudsen.zip

I TILLEGG: I denne obligen er det fint hvis dere laster opp **alle .java filene en og en** direkte, i *tillegg* til en .zip fil med alle.

Teori

Oppgave 1.1

Lag deg en oversikt over hva følgende ord/begreper betyr, *med egne ord* (lurt å gjøre dette bra nå, kan være nyttig til eksamen):

- Class
- Object (konseptet, ikke den innebygde Java-klassen)
- Instansvariabel
- Overloading
- Overriding
- Extends
- Polymorphism
- private,public,(protected) (klasse,variabel,metode)
- this og super

Oppgave 1.2

Hva skjer når vi kompilerer et java-program? Hvorfor må vi gjøre dette?

Programmering

Vi skal lage en applikasjon som skal holde oversikt over digital media, i form av filmer og tvserier. I denne digitale hverdagen vi befinner oss i kan vi ikke stole på noen, vi ønsker å generere og vedlikeholde alle dataene helt selv. Vi begynner med å fokusere på tvserier.

Obligen er strukturert opp i oppgaver, men dere skal levere ett endelig prosjekt med sluttresultatet av alle oppgavene.

Oppgave 2.1 - Modellklasser

Vi skal kunne opprette TVSerier med tilhørende data. Vi trenger derfor noen klasser for å representere dette.

Lag derfor 2 klasser med instansvariabler:

- **Episode**
  - tittel
  - episodeNr
  - sesongNr
  - spilletid
- **TVSerie**
  - tittel
  - beskrivelse
  - utgivelsesdato
  - episoder **(spoiler: bruk ArrayList<Episode>)**

Hint, hint: Merk at instansvariabelen episoder i **TVSerie**-klassen er skrevet i flertall. Hva for slags "datatype" kan være relevant for dette? Dere står fritt til å velge fornuftige datatyper selv, men et lite tips i forhold til utgivelsesdato (da Java har flere forskjellige klasser for håndtering av dette), er å benytte LocalDate (https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/time/LocalDate.html) .

Vi ønsker to konstruktører for å kunne opprette en **Episode**, en med alle instansevariablene, og en med alle unntatt spilletid (overloading).

En episode skal kunne legges til enkeltvis via en metode *leggTilEpisode(Episode episoden)* i **TVSerie**-klassen.

Husk å gjøre instansvariablene *private*, og lag get- og set-metoder for disse (innkapsling).

Oppgave 2.2 - Opprette objekter

Lag en kjørbar klasse kalt *Main.java* som skal benytte klassene du akkurat laget.

Opprett et objekt av **TVSerie**-klassen for en TVSerie du selv ønsker. Opprett og legg til noen **Episode**-objekter til denne TVSerien.

Oppgave 2.3 - Utskrift og toString()

Implementer toString()-metoden i klassene du laget i Oppgave 2.1, lag en passende utskrift som gir relevant informasjon om objektet.

Benytt denne til å skrive ut informasjon om tvserien og noen av episodene du har laget i *Main.java*.

Oppgave 2.4 - Hente episoder

Vi ønsker å kunne hente ut alle episoder for en spesifikk sesong.

Lag en metode i TVSerie som henter ut alle episodene i en sesong (returnerer en liste med episoder for den sesongen man ønsker).

public ArrayList<Episode> hentEpisoderISesong(int sesong)

For å teste at denne metoden fungerer som den skal, ønsker vi å fylle opp et TVSerie-objekt med Episode-objekter som går over flere sesonger. I Main.java skal du legge til 5 sesonger med 20 episoder. Det holder med generisk informasjon om hver episode (i.e. "Episode 14"). (hint: bruk løkker)

Test ut den nye metoden i Main.java og hent alle episoder i sesong 4 og skriv de *deretter* ut til konsollen.

Oppgave 2.5 - Gjennomsnittlig spilletid

Vi ønsker å holde oversikt over gjennomsnittlig spilletid for alle episoder i en TvSerie.

Lag en instansvariabel *gjennomsnittligSpilletid* i TVSerie. Lag kun en "getter" for denne.
Lag så en private metode som heter *oppdaterGjennomsnittligSpilletid()*, denne skal kalles hver gang det legges til en ny episode.
Metoden skal regne ut gjennomsnittlig spilletid og oppdatere den tilhørende instansvariablen.

Oppgave 2.6 - Generere tilfeldig spilletid

Benytt klassen **java.util.Random** (https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/util/Random.html) til å generere et tall for spilletid mellom 20 og 30 for alle episodene du opprettet i oppgave 2.4. Hent så ut gjennomsnittligSpilletid og skriv den ut til terminalen.

Oppgave 2.7 - Holde på antall sesonger

Vi vil kun kunne legge til nye episoder som er i en sesong som eksisterer, eller 1 høyere enn den sesongen vi er i nå. Så du skal f.eks. *ikke* kunne legge til episoder i sesong 5, hvis det bare er registrert episoder i sesong 1 og 2 så langt. Men det vil være greit å begynne å registere de i sesong 3.

Legg til en instansvariabel antallSesonger i TVSerie. Lag kun en "getter" for denne.

I metoden *leggTilEpisode()* skal du:
Sjekke om episoden tilhører en sesong som ikke er mer enn én høyere enn gjeldende "antallSesonger". Hvis

det er tilfellet, skriv ut en feilmelding til konsollen og legg ikke til episoden.
Oppdater "antallSesonger" hvis episoden sin sesong er én høyere.

Oppgave 2.8 - "Teste" antall sesonger

Test implementasjonen fra forrige oppgave i Main.java.

Se at du får feilmelding om du prøver å legge til en episode i en sesong som er for høy.

Se at antallSesonger oppdaterer seg korrekt ved scenariet over (og ikke i andre tilfeller).