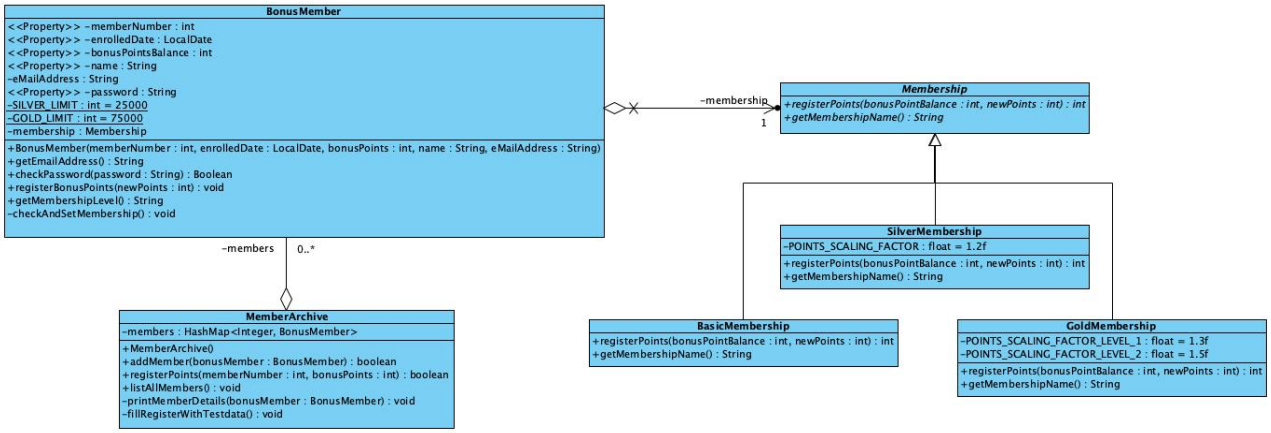
Arv og polymorfi

Problemstilling

Følgende klassediagram er gitt:



Figur 1 Klassediagram for bonus ordning

Et flyselskap tilbyr tre ulike typer bonuskort. En kunde tjener opp poeng hver gang han reiser med fly. Når en kunde melder seg inn i ordningen starter han alltid som medlem på nivå *Basic*.

Dersom kunden reiser ofte og tjener opp mange poeng det første året han er medlem i ordningen oppgraderes han automatisk til nivå *Sølv* eller *Gull*.

For å bli sølvmedlem må kunden tjene opp 25.000 poeng i løpet av ett år. Kravet til gullmedlemskap er 75.000 poeng.

En kunde er enten **basic**-medlem, **sølv**-medlem *eller* **gull**-medlem.

Alle medlemmer tjener i utgangspunktet det samme for den samme reisen på samme klasse. (Og her stopper likheten mellom oppgaveteksten og velkjente bonusordninger...). Sølvmedlemmer får et påslag i antall poeng på 20%, mens gullmedlemmer får et påslag på 30% på poeng fra og med 75.000 til 90.000, og 50% for poeng over (fra og med) 90.000.

Eksempel: Dersom en reise gir 5.000 poeng i bonus, skal:

- Et basic-medlem få 5.000 poeng lagt til sin saldo
- Et sølv-medlem få 5.000 x 1.2 = 6.000 poeng lagt til sin saldo
- Et gull-medlem få 5.000 x 1.3 = 6.500 poeng dersom hans bonussaldo er < 90.000, og 5.000 x 1.5 = 7.500 poeng dersom hans bonussaldo er >= 90.000.

Forenklinger i denne oppgaven:

- Vi lagrer ikke enkelttransaksjoner, bare poengsaldoen.
- Vi ser bort fra opptjening i løpet av et år (siden vi ikke holder oversikt over hver poengregistrering).

Versjonskontroll

Dette er en ypperlig oppgave å trene mer på versjonskontroll samtidig som du løser oppgaven. Opprett et *lokalt repository* i prosjektmappen, og legg deg til gode vaner med å *sjekke inn* (commit) endringer i koden din fortløpende. Husk gode kommentarer til innsjekking.

Reflekter også over hvilke filer og mapper i prosjektet du bør legge under versjonskontroll, og hvilke du bør la versjonskontrollsystemet (Git) ***ignorere***.

Prøv gjerne også å lag deg et *remote repository* (i GitHub eller GitLab) og tren på «*push*» til sentral repo.

For studenter i Ålesund: Dere vil få tilgang til GitHub Classroom i denne øvingen på tilsvarende måte som i øving 1.

Maven

I denne oppgaven skal du opprette prosjektet som et **Maven prosjekt** i din foretrukne IDE. Sett prosjektet til JDK versjon 8 eller 11 (som er Long Term Support (LTS)), og legg til biblioteket for JUnit5 (ved å legge til groupID: **`org.junit.jupiter`**, artifactID: **`junit-jupiter`** og versjon: **5.7.0**).

Filer som blir utlevert

Du får følgende filer utlevert som du kan bruke i løsningen din i oppgave 4:

- **MemberArchive.java** – klassen og skallet til metodene er gitt. Resten skal du fylle ut.
- **MemberArchiveClient.java** – en enkel klientklasse med metoden *public static void main()* implementert som utfører en enkel test av løsningen din.

Oppgave 1

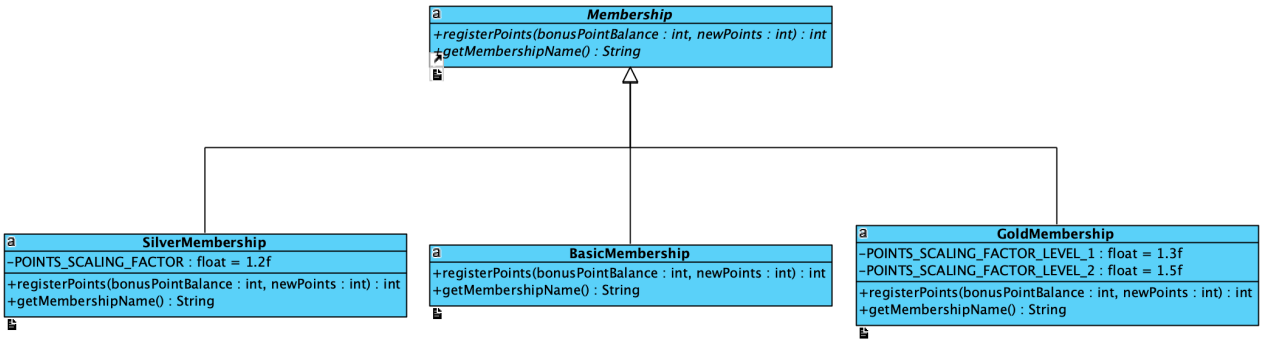
Du skal i denne oppgaven konsentrere deg om alle klassene i Figur 1.

Du skal programmere klassene **BonusMember**, **Membership**, **BasicMembership**, **SilverMembership** og **GoldMembership**.

Start med Membership-klassene.

Klassen Membership med subklasser

Denne klassen er en ***abstrakt superklasse*** felles for alle klasser som representerer et medlemskapsnivå. Hvert nivå implementeres da som **subklasser** til denne superklassen, som vist i klassediagrammet under.



Figur 2 Klassediagram som viser Membership-klassene

Som det fremgår av diagrammet, har superklassen Membership to **abstrakte metoder**:

- **`int registerPoints(int bonusPointBalance, int newPoints)`** – metoden tar med inn som parameter gjeldende bonuspoeng saldo, og poengene som skal registreres. Metoden beregner så ny saldo basert på fordelene i gitt medlemskap. Den nye saldoen blir så returnert.
- **`String getMembershipName()`** – returnerer en tekststreng med «navnet» til medlemskapet. Er medlemskapet et basic medlemskap, returneres teksten «Basic», er det sølv, returneres teksten «Silver» osv.

Basert på klassediagrammet over, implementer alle 4 klassene. Husk at de tre subklassene **må** redefinere (override) de to abstrakte metodene i superklassen.

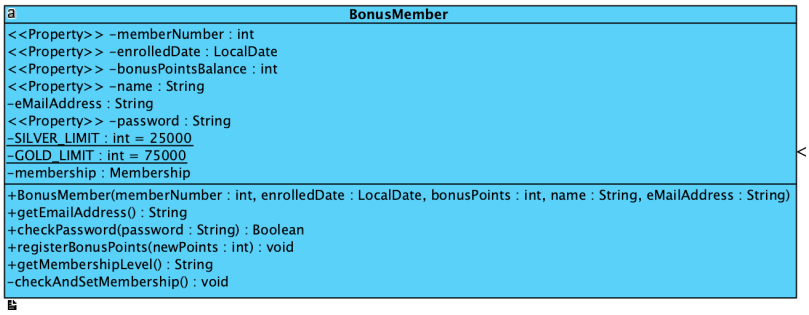
Faktorene som skal benyttes for å beregne bonuspoeng basert på medlemskapsnivå er definert som konstanter i respektive klasser. Merk at disse er av datatypen ***float***, så når dere skal utføre beregningen av ny saldo (som er datatypen ***int***), kan det være nyttig å benytte seg av følgende metode i matematikk-biblioteket (i klassen **Math**):

```
int Math.round(float value)
```

(Se Javadoc: https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html#round-float-)

Oppgave 2

Klassen BonusMember



Figur 3 Klassen BonusMember

Som du ser av UML-spesifikasjonen av klassen **BonusMember,** så har den følgende objektvariabler/felt:

```
private int memberNumber;
private LocalDate enrolledDate;
private int bonusPointsBalance = 0;
private String name;
private String eMailAddress;
private String password;

private Membership membership;
```

Klassen **LocalDate** ligger i pakken java.time (https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html).

I tillegg til feltene er det definert to *konstanter*:

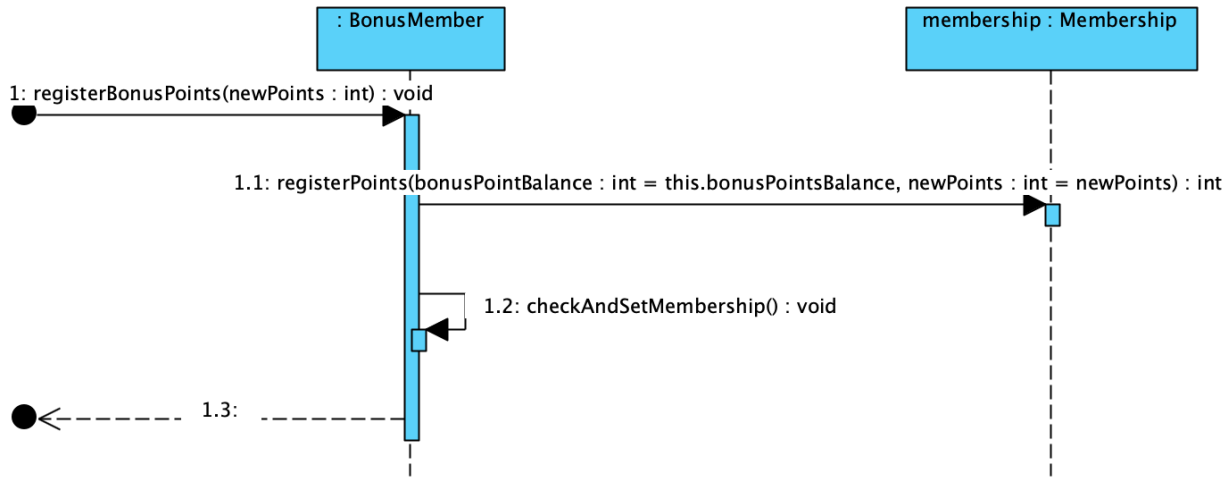
```
private static final int SILVER_LIMIT = 25000;
private static final int GOLD_LIMIT = 75000;
```

Disse konstantene brukes for å avgjøre hvilket nivå på medlemskapet BonusMember har gjort seg fortjent til.

- Implementer samtlige **aksessormetoder** for feltene til klassen.
- Implementer så metoden **checkPassword(String password)**. Denne metoden skal returnere **true** dersom passordet i parameteren *password* matcher passordet til medlemmet. **False** hvis ikke.

Metoden registerBonusPoints()

Denne metoden registrerer nye bonus poeng for medlemmet. Fordi bonuspoengene skal vektes ulikt basert på medlemskap, ***deligeres*** registreringen til *Membership*-objektet. Sekvensdiagrammet under viser hvordan:



Figur 4 Sekvensdiagram av metoden registerBonusPoints()

Implementer metoden registerBonuspoint(). Denne metoden gjør også bruk av en «hjelpemetode»; **checkAndSetMembership()** som, basert på gjeldende bonuspoeng saldo, setter riktig medlemskapsnivå for medlemmet. Hvordan vil du implementere denne metoden? (Det kan også være lurt å kalle denne metoden fra **konstruktøren** til klassen for å sette initielt medlemskapsnivå.)

Fullfør resten av metodene i klassen.

Oppgave 3

Dersom du har greit med tid igjen når du er kommet hit, gå til Oppgave 4.

Dersom du sliter med å komme i mål innen fristen, gjør følgende alternativ til Oppgave 4:

Lag en enkel klientklasse (kall den gjerne **SimpleClient**), med metoden *public static void main(String[] args)*. Legg inn noen kodelinjer i main()-metoden som:

1. Oppretter et bonusmedlem med en initiell bonuspoeng saldo.
2. Skriv ut medlemmets detaljer (minimum medlemsnummer, navn, bonuspoeng og hvilket medlemskapsnivå medlemmet har).
3. Registrer nye bonuspoeng på medlemmet, f.eks. tilstrekkelig til at medlemmet skifter medlemskapsnivå.
4. Skriv ut medlemmets detaljer igjen, og kontroller om medlemmet a) har fått utregnet riktig ny saldo, og b) har fått riktig medlemskapsnivå.

Oppgave 4 – Hvis du rekker..

I denne oppgaven skal du programmere metodene i klassen **MemberArchive**.

Følgende gjelder:

- **findPoints()** - skal ta medlemsnummer og passord som argument og returnere antall poeng denne kunden har spart opp. Returner en negativ verdi hvis medlem med dette nummeret ikke fins, eller passord er ugyldig.
- **registerPoints ()** - skal ta medlemsnummer og antall poeng som argument og sørge for at riktig antall poeng blir registrert for dette medlemmet. Returner false dersom medlem med dette nummeret ikke fins.
- **addMember()** - skal ha følgende signatur:

```
public int addMember(BonusMember bonusMember)
```

- **listAllMembers()** - skal liste alle medlemmene i arkivet. Tips: lurt å også lage en egen metode for å skrive ut detaljene til et medlem ;-)

Til slutt skal du lage en metode som fyller arkivet ditt med 5 medlemmer, med medlemsnr 1 – 5.

Test løsningen din ved å kjøre main()-metoden i MemberArchiveClient-klassen. Legg spesielt merke til at medlemmer endrer medlemskapsnivå riktig basert på tildelte bonuspoeng.

Har du fortsatt tid til overs, kan du opprette en del JUnit-tester for å teste at logikken med medlemskapsnivå fungerer riktig. Husk da både **positive** og **negative** tester ☺

LYKKE TIL ☺