NTNU

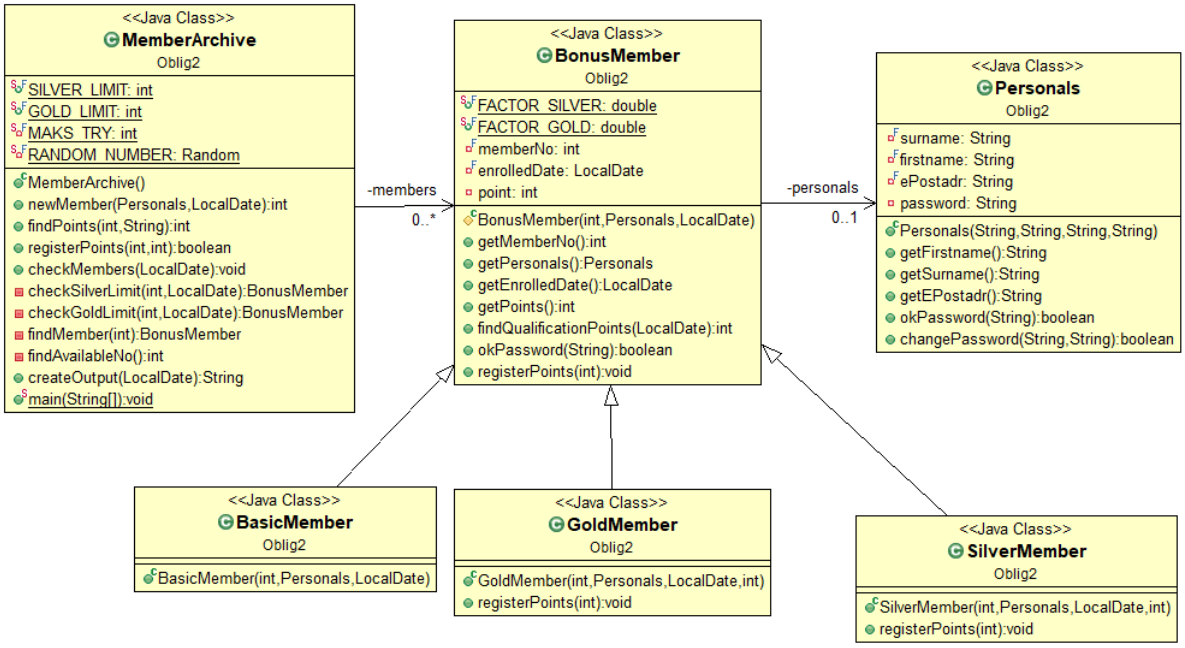
Oblig 2

IDATT2001 Programmering 2
Institutt for datateknologi og informatikk (IDI), NTNU

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 25000 poeng i løpet av ett år. Kravet til gullmedlemskap er 75000 poeng. En kunde er *enten* basic-medlem, sølvmedlem *eller* gullmedlem.

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 50%. Eksempel: Dersom en tur gir 5000 poeng i gevinst, skal 5000 poeng registreres for

basic-medlemmer. Sølvmedlemmer skal få 5000 x 1.2 = 6000 poeng, mens gullmedlemmer får 5000 x 1.5 = 7500 poeng.
Forenklinger i denne oppgaven:

- Vi lagrer ikke enkelttransaksjoner, bare poengsaldoen.
- For å kunne oppgraderes til sølv- eller gullmedlem, må kunden tjene henholdsvis 25000 og 75000 poeng *det første året* han/hun er medlem.

Oppgave 1

Du skal i denne oppgaven konsentrere deg om alle klassene på figur 1 *unntatt* klassen MemberArchive.

Klassen Persoanls er gitt (Se vedlegg).

Du skal programmer klassene BonusMember, BasicMember, SilverMember og GoldMember.

Klassen BonusMember har følgende objektvariabler:

```
private final int memberNo;
private final Personals personals;
private final LocalDate enrolledDate;
private int point = 0;
```

Klassen LocalDate ligger i pakken java.time.

Utfyllende forklaring til operasjonene i klassen BonusMember:
De fire første operasjonene
(**getMemberNo()**, **getPersonals()**, **getEnrolledDate()** og **getPoints()**) programmeres som vanlige get-metoder.

- **findQualificationPoints()** skal returnere antall poeng som kan kvalifisere til oppgradering av medlemskapet til sølv eller gull. Dersom innmeldingsdatoen ligger mindre enn 365 dager bak i tid i forhold til datoen som sendes inn som argument, returneres antall poeng. Hvis det er mer enn ett år siden kunden meldte seg inn, returneres 0 poeng. Du kan finne differansen mellom to objekter av klasse LocalDate på denne måten:

  ```
  long dagerMellom = ChronoUnit.DAYS.between(enrolledDate, dato);
  ```

- **okPassword()** tar et passord som argument, og returnerer true dersom det er ok.
- **registerPoints()** skal ta antall poeng som argument og registrere disse i henhold til reglene foran. (Du skal ikke tenke på oppgradering til gull- og sølvmedlemskap her.)

Her følger noen testdata (findQualificationPoints() i forhold til datoen 10.02.08):

Person	Type	Startpoeng	Innmeldt dato	poeng*)	findQualificationPoints ()	findPoints()	poeng	findQualificationPoints ()	findPoints()
Ole	Basic	0	15.02.06	30.000	0	30.000	15.000	0	45.000
Tove	Basic	0	05.03.07	30.000	30.000	30.000			
Tove	Sølv	30.000	05.01.07	50.000	90.000	90.000			
Tove	Gull	90.000	10.08.07	30.000	135.000	135.000			

*) poeng uten påslag for sølv-/gullmedlemskap
Testdataene er lagt inn i TestBonusMember (Se vedlegg). Du kan bruke det ved testing - vurder om det er vesentlige ting som ikke blir testet.

Oppgave 2

Du skal nå 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 nr 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 nr ikke fins.
newMember() skal ha følgende metodehode:

```
public boolean registerPoints(int no, int number)
```

Metoden skal opprette et objekt av klassen BasicMember og legge dette inn i arkivet. (Alle medlemmer begynner som basic-medlemmer.) Metoden skal returnere medlemsnummeret.

Metoden skal bestemme medlemsnummeret ved å kalle følgende private metode:

```
private int findAvailableNo()
```

Denne metoden skal hente ut et tilfeldig heltall (bruk klassen Random) som ikke allerede er i bruk som medlemsnr.

checkMembers() skal gå gjennom alle medlemmene og foreta oppgradering av medlemmer som er kvalifisert for det. Basic-medlemmer kan kvalifisere seg for sølv eller gull, mens sølvmedlemmer kan kvalifisere seg for gull. *Tips*: Du trenger å finne ut hvilken klasse et objekt tilhører. Bruk operatoren instanceof. Det er ikke mulig å omforme klassetilhørigheten til et objekt. Du må i stedet lage et nytt objekt med data fra det gamle. Det nye objektet må legges inn i ArrayListen på den plassen der det gamle lå (bruk metoden set()).

Lag en enkel testklient der spesielt metoden checkMembers() blir prøvd ut. Du kan finne det hensiktsmessig å lage flere metoder i klassen Medlemsarkiv for å få testet tilstrekkelig. Hvis du trenger å skrive ut hvilken klasse et objekt tilhører, kan du bruke metoden getClass() i klassen Object.

```
package Oblig2;

/**
 * Personalia.java  2010-01-18
 *
 * Klasse med personopplysninger: fornavn, etternavn, epostadresse og passord.
 * Passordet kan endres, men da mÃ¥ det nye vÃ¦re forskjellig fra det gamle.
 * Passordkontrollen skiller ikke mellom store og smÃ¥ bokstaver.
 */
class Personals {
  private final String surname;
  private final String firstname;
  private final String ePostadr;
  private String password;

  /**
   * KonstruktÃ¸r:
   * Alle data mÃ¥ oppgis: fornavn, etternavn, ePostadr, passord
   * Ingen av dataene kan vÃ¦re null eller blanke strenger.
   */
  public Personals(String firstname, String surname, String ePostadr, String password) {
    if (firstname == null || surname == null || ePostadr == null || password == null ||
        firstname.trim().equals("") || surname.trim().equals("") ||
ePostadr.trim().equals("") || password.trim().equals("")) {
         throw new IllegalArgumentException("Et eller flere konstruktrør argumenter er
null og/eller blanke.");
    }
    this.firstname = firstname.trim();
    this.surname = surname.trim();
    this.ePostadr = ePostadr.trim();
    this.password = password.trim();
  }

  public String getFirstname() {
    return firstname;
  }

  public String getSurname() {
    return surname;
  }

  public String getEPostadr() {
    return ePostadr;
  }

  /**
   * Metoden returnerer true dersom passordet er korrekt.
   * Passordkontrollen skiller ikke mellom store og smÃ¥ bokstaver.
   */
  public boolean okPassword(String password) {
    return this.password.equalsIgnoreCase(password);
  }

  /**
   * Metoden setter nytt passord, dersom det er forskjellig fra
   * det gamle. To passord betraktes som like dersom det kun er
   * forskjeller i store/smÃ¥ bokstaver.
   *
   * Metoden returnerer true dersom passordet ble endret, ellers false.
   */
```

```
  public boolean changePassword(String oldPassword, String newPassword) {
    if (oldPassword == null || newPassword == null) {
      return false;
    }
    if (!password.equalsIgnoreCase(oldPassword.trim())) {
      return false;
    } else {
      password = newPassword.trim();
      return true;
    }
  }
}
```

```
package Oblig2;

import java.time.LocalDate;

class TestBonusMember {
  public static void main(String[] args) throws Exception {
    Personals ole = new Personals("Olsen", "Ole", "ole.olsen@dot.com", "ole");
    Personals tove = new Personals("Hansen", "Tove", "tove.hansen@dot.com", "tove");
    LocalDate testdato = LocalDate.of(2008, 2, 10);
    System.out.println("Totalt antall tester: 8");

    BasicMember b1 = new BasicMember(100, ole, LocalDate.of(2006, 2, 15));
    b1.registerPoints(30000);
    if (b1.findQualificationPoints(testdato) == 0 && b1.getPoints() == 30000) {
      System.out.println("Test 1 ok");
    }
    b1.registerPoints(15000);
    if (b1.findQualificationPoints(testdato) == 0 && b1.getPoints() == 45000) {
      System.out.println("Test 2 ok");
    }

    BasicMember b2 = new BasicMember(110, tove, LocalDate.of(2007, 3, 5));
    b2.registerPoints(30000);
    if (b2.findQualificationPoints(testdato) == 30000 && b2.getPoints() == 30000) {
      System.out.println("Test 3 ok");
    }

    SilverMember b3 = new SilverMember(b2.getMemberNo(), b2.getPersonals(),
b2.getEnrolledDate(), b2.getPoints());
    b3.registerPoints(50000);
    if (b3.findQualificationPoints(testdato) == 90000 && b3.getPoints() == 90000) {
      System.out.println("Test 4 ok");
    }

    GoldMember b4 = new GoldMember(b3.getMemberNo(), b3.getPersonals(),
b3.getEnrolledDate(), b3.getPoints());
    b4.registerPoints(30000);
    if (b4.findQualificationPoints(testdato) == 135000 && b4.getPoints() == 135000) {
      System.out.println("Test 5 ok");
    }

    testdato = LocalDate.of(2008, 12, 10);
    if (b4.findQualificationPoints(testdato) == 0 && b4.getPoints() == 135000) {
      System.out.println("Test 6 ok");
    }

    if (!ole.okPassword("OOO")) {
      System.out.println("Test 7 ok");
    }
    if (tove.okPassword("tove")) {
      System.out.println("Test 8 ok");
    }
  }
}
```

```java
package Oblig2;

import java.time.LocalDate;
import java.time.Period;
import java.time.ZoneId;
import java.time.temporal.ChronoUnit;
import java.util.Date;

import static lib.Out.*;


/**
 * @author rouhani
 * I denne klassen tester vi 3 forskjellige måter å finne antall dager mellom to datoer
 * Vi ser at doTestPeriod er feil og kan ikke brukes for å finne denne differansen.
 */
public class TestPeriod {
	private void doTestPeriod(LocalDate date1, LocalDate date2) {

		int daysBetween = Period.between(date1, date2).getDays();

		out("Test med doTestPeriod: "+daysBetween);
	}

	private void doTestChronoUnit(LocalDate date1, LocalDate date2) {
		long daysBetween = ChronoUnit.DAYS.between(date1, date2);

		out("Test med doTestChronoUnit: "+daysBetween);
	}

	private void doTestDays(LocalDate date1, LocalDate date2) {
		ZoneId defaultZoneId = ZoneId.systemDefault();

		Date d1 = Date.from(date1.atStartOfDay(defaultZoneId).toInstant());
		Date d2 = Date.from(date2.atStartOfDay(defaultZoneId).toInstant());

		long daysBetween=(d2.getTime()-d1.getTime())/86400000;

		out("Test med doTestDays: "+daysBetween);
	}


	public static void main(String[] args) {
		TestPeriod t = new TestPeriod();
		LocalDate date1;
		LocalDate date2;

		out("\nTest 1: OK");
		date1 = LocalDate.of(2020, 2, 5);
		date2 = LocalDate.of(2020, 2, 10);

		t.doTestPeriod(date1, date2);
		t.doTestChronoUnit(date1, date2);
		t.doTestDays(date1, date2);


		out("\nTest 2: Feil på doTestPeriod");
		date1 = LocalDate.of(2020, 2, 5);
		date2 = LocalDate.of(2020, 3, 10);
```

```
            t.doTestPeriod(date1, date2);
            t.doTestChronoUnit(date1, date2);
            t.doTestDays(date1, date2);


            out("\nTest 3: Feil på doTestPeriod");
            date1 = LocalDate.of(2019, 2, 5);
            date2 = LocalDate.of(2020, 2, 10);

            t.doTestPeriod(date1, date2);
            t.doTestChronoUnit(date1, date2);
            t.doTestDays(date1, date2);
        }
}
```