Høgskolen i Østfold

EKSAMEN

Emnekode: ITF10611	Emne: Objektorientert Programmering
Dato: 19 mai 2017	Eksamenstid: kl. 09:00 til kl. 13:00

Hjelpemidler: To A4-ark (fire sider) med egne notater	Faglærer: Per Bisseberg

Eksamensoppgaven:
Oppgavesettet består av 18 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 presist.

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.

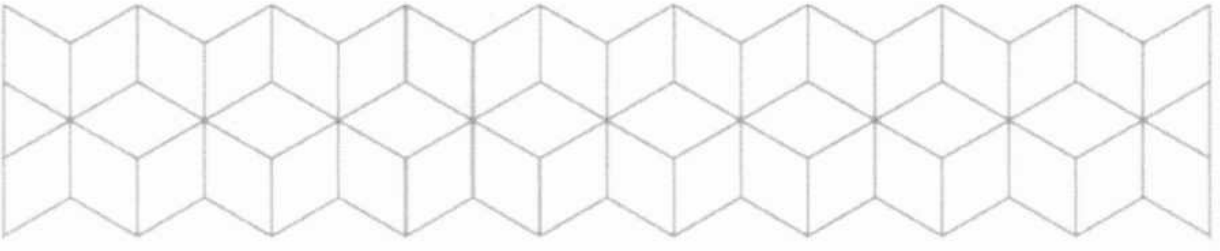
Pass også på å svare på alle oppgaver. Dersom du står fast er det bedre å skrive litt i grove trekk hvordan du ser for deg at oppgaven kan løses, enn å ikke skrive noe i det hele tatt.

Ønsker dere alle en riktig god sommer og lykke til i videre studier. Takk for et virkelig hyggelig siste semester :-)

Lykke til!

Sensurdato: 14. juni 2017

Karakterene er tilgjengelige for studenter i Studentweb senest 2 virkedager etter oppgitt sensurfrist.
www.hiof.no/studentweb



Oppgave 1 (25 %)

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 ved kode og/eller illustrasjoner.

Oppgave 1.1

a) Forklar begrepet innkapsling og eksemplifiser med enkel kode.
b) Forklar begrepet objektreferanse og eksemplifiser med enkel kode

Oppgave 1.2

Forklar disse nøkkelordene fra Java

a) static
b) new
c) instanceof

Oppgave 1.3

Gjør rede for begrepet «arv». Hva er det? Hvordan benyttes det? Hvorfor benytter vi det? Hva er effektene av det? Underbygg utredningen med kode og UML.

Oppgave 2 (25 %)

Analyse av kode.

Oppgave 2.1 (10 %)

Disse oppgavene inneholder en eller flere feil som gjør at koden ikke kompilerer. Du skal skrive hva kompileringsfeilen(e) er og skrive koden som skal til for at koden skal kompilere.

Oppgave 2.1.1

```
public abstract class A {
	private String navn;

	public A(String navn) {
		this.navn = navn;
	}
}

public class B extends A{
	private int verdi;

	public B(String navn, int verdi) {
		this.navn = navn;
		this.verdi = verdi;
	}
}
```

Oppgave 2.1.2

```
public interface  A {

     public void hentInformasjon() {
          System.out.println(navn + ": " + verdi);
     };

}

public class B implements A{
     private String navn;
     private int verdi;

     public B(int verdi, String navn) {
          this.verdi = verdi;
          this.navn = navn;
     }
}
```

Oppgave 2.2 (15 %)

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

Oppgave 2.2.1

```
public class Bil {
	private String merke;
	private String modell;
	private String aarsmod;

	public Bil(String merke, String modell, String aarsmod) {
		this.merke = merke;
		modell = modell;
		aarsmod = aarsmod;
	}

	@Override
	public String toString() {
		return aarsmod + ": " + merke + " " + modell;
	}

}

// main metode
public static void main(String[] args) {
	ArrayList<Bil> biler = new ArrayList<>();

	biler.add( new Bil("Volvo", "XC90", "2015") );
	biler.add( new Bil("BMV", "X5", "2012") );
	biler.add( new Bil("Audi", "A8", "2013") );

	for(Bil b: biler) {
		System.out.println(b);
	}

}
```

Oppgave 2.2.2

```
public class Kunde {
	private String fnavn;
	private String enavn;
	private int alder;
	private boolean myndig = false;

	public Kunde(String fnavn, String enavn, int alder) {
		this.fnavn = fnavn;
		this.enavn = enavn;
		this.alder = alder;
		if(alder >= 18) {
			myndig = true;
		}
	}

	@Override
	public String toString() {
		if(myndig) {
			return fnavn + " " + enavn;
		}
		return fnavn + " " + enavn + ", " + alder;
	}

	public void setAlder(int alder) {
		if(alder >= 18) {
			myndig = true;
		}
		this.alder = alder;
	}

	// øvrig get og set ikke tatt med, men finnes for alle felt.
}

// main metode
public static void main(String[] args) {
	ArrayList<Kunde> kunder = new ArrayList<Kunde>();

	Kunde k1 = new Kunde("Pål", "Kork", 15);
	Kunde k2 = k1;
	Kunde k3 = new Kunde("Kari", "Johansen", 19);

	kunder.add(k1);
	kunder.add(k2);
	kunder.add(k3);

	k2.setAlder(18);

	for(Kunde k: kunder){
		System.out.println(k);
	}
}
```

Oppgave 2.2.3

Vi benytter Kunde-klassen fra forrige oppgave

```
// main metode
public static void main(String[] args) {
	ArrayList<Kunde> kunder = new ArrayList<Kunde>();

	kunder.add( new Kunde("Pål", "Kork", 15) );
	kunder.add( new Kunde("Hanne", "Flaske Vin", 21) );
	kunder.add( new Kunde("Kari", "Zitròn Bruce", 17) );
	kunder.add( new Kunde("Ole", "Riesling", 23) );

	Collections.sort(kunder, new Comparator<Kunde>() {

		@Override
		public int compare(Kunde k1, Kunde k2) {
			if(k1.getMyndig() == k2.getMyndig()) {
				return 0;
			} else if(k1.getMyndig()){
				return -1;
			} else {
				return 1;
			}
		}
	});

	for(Kunde k: kunder){
		System.out.println(k);
	}
}
```

Oppgave 3 (50 %)

Vi skal utvikle en prototype av et system for en oppdragsgiver, alarmselskapet «***Falsk Alarm***». Vi har nettopp begynt utviklingen og vi jobber med å få på plass de første delene av kjernefunksjonalitet. Vi trenger ikke tenke på persistent lagring på dette tidspunktet, du kan anta at dette allerede er på plass slik at systemet til enhver tid vil holde på lagret informasjon. Vedlagt finner du de klasser vi har så langt.

Falsk Alarm trenger et system som:

- Registrerer alarminstallasjoner med tilhørende alarmpunkt.
- Registrerer eiere og kontaktpersoner for en installasjon
- Registrerer ansatte i vaktselskapet
- Registrerer alarmer som utløses hos installasjonene
- Registrerer utrykninger til varslede alarmer og hvilke ansatte som rykker ut

Elementene i systemet er som følger:

- **Main**
  - **Denne klassen viser et typisk scenario systemet skal støtte, les denne og prøv å få en oversikt om hvordan det henger sammen**.
- Person
  - Alle personer registreres med diverse personalia og kontaktinformasjon
- Eier
  - En **Person** som er eier av en installasjon, altså alarmselskapets kunde. Registreres med kundenummer.
- Kontakt
  - En **Person** som skal kunne kontaktes når alarm utløses for å sjekke om alarmen er reell.
- Ansatt
  - En **Person** som er ansatt i alarmselskapet. Registreres med rollebeskrivelse, f.eks. «vekter».
- Installasjon
  - En alarminstallasjon i ett fysisk bygg. Registreres med id til alarmsentral, hvilken person som er **Eier** og evt. hvilke personer som er **Kontakt**personer for denne installasjonen.
- Alarmpunkt
  - Et komponent som er tilknyttet **Installasjon**ens alarmsentral. For enkelhetsskyld skiller vi ikke på dette tidspunktet på ulike typer alarmkomponent utover en String-verdi som beskriver typen. F.eks.: «Røykdetektor», «Bevegelsessensor» etc.
- Alarm
  - En alarm opprettes når et/eller flere alarmpunkt i en installasjon utløses. Registreres med hvilken Installasjon som alarmerer og en rapport om hvilke alarmer som er utløst.
- Utrykning
  - En utrykning gjøres mot en spesifikk **Alarm** og registreres med hvilke **Ansatt**(e) som er med på utrykningen.
- Kontaktbar
  - Er et grensesnitt/interface som kun benyttes til å markere ulike typer personer som kan være kontaktpersoner. Så langt er dette begrenset til **Eier** og **Kontakt**

Oppgave 3.1

Lag et klassediagram, det er ikke nødvendig å ta med felter og metoder – **kun klassenavn/interfacenavn**, over alle elementene i systemet hvor du tydelig får frem eventuelle arverelasjoner, implementasjoner og ulike typer assosiasjoner. Du trenger ikke tegne inn klassen Main. Du trenger heller ikke å tegne nye elementer eller assosiasjoner som evt. dukker opp i systemet i senere oppgaver.

Oppgave 3.2

a) Skriv ferdig konstruktørene i klassene **Ansatt**, **Eier** og **Kontakt**. Alle felter/«klassevariabler» skal settes i konstruktøren.

b) I klassen **Installasjon** har vi metoden ***lagAlarmRapport ()***. Denne metoden skal returnere en tekststreng som gir en oversikt over hvilke **Alarmpunkt** som er utløst.
Denne skal f.eks. se slik ut:

```
Alarm utløst:
Sentral: SN1239A_12801
Lokasjon: Parkvn. 8 1768 Halden
Utløste alarmpunkt:
Røykdetektor: Leilighet 1 - Soverom 1
Røykdetektor: Felles - Trappegang
```

Tips: «\n» vil lage linjeskift i en String.

Oppgave 3.3

Klassen **Installasjon** definerer metoden ***testAvInstallasjon().*** Denne metoden skal benyttes til test av installasjonene og tilhørende alarmpunkter. Implementer logikk i denne metoden som oppretter og returnerer en **Alarm** med informasjon om hvilken **Installasjon** og hvilke **Alarmpunkt(er)** denne alarmen gjelder. Et scenario for bruk av denne metoden kan du se i klassen **Main**.

Oppgave 3.4

Lag en metode i klassen **Installasjon** som lar oss registrere hvilke kontaktbare personer denne installasjonen har. Vi ønsker ikke at man skal kunne registrere den samme personen som kontaktperson flere ganger på samme Installasjon så dette må du ta høyde for.

Oppgave 3.5

Gjør tillegg/forandring i klassen **Installasjon** som definerer at den naturlige sorterings-ordenen på Installasjonsobjekt er stigende på postnummer.

Oppgave 3.6

Vi har fått et ønske fra alarmselskapet om å legge til funksjonalitet som lar dem registrere hvilke oppgaver som har vært utført på en gitt **Utrykning,** slik at de kan legge ved denne informasjonen i faktura (du trenger ikke implementere funksjonalitet for fakturering) til kunden. Registrering av disse oppgavene skal gjøres via utrykningsobjektet. En oppgave skal inneholde dato/tid for oppstart og avslutning samt en tekstlig beskrivelse av hva som er utført.

F.eks.: På utrykning til alarm ved installasjon i Bankegaten 12 1776 Halden der alarmpunkt i form av en magnetkontakt til ytterdøren er utløst. Her blir følgende oppgaver utført:

- «Kontaktet kontaktpersoner, men ingen respons», start: 19.05.2017:23:12 – slutt: 19.05.2017:23:14
- «Utvendig inspeksjon av utløst alarmpunkt, ingen synlige tegn til innbrudd», start: 19.05.2017:23:21 – slutt: 19.05.2017:23:32
- «Opprettet kontakt med beboer, Edgar Utenström. Beboer er åpenbart forvirret. Konklusjon: falsk alarm», start: 19.05.2017:23:36 – slutt: 19.05.2017:23:52

P.S: Du trenger ikke tenke på formatering av dato/tid. Se vedlegg for GregorianCalendar dokumentasjon.

Dette medfører en viss omstrukturering av systemarkitekturen. Gjør rede for hvordan du ville løst dette. Benytt UML (du trenger ikke tegne hele systemet på nytt, kun de delene som påvirkes av omstruktureringen), kode for nye systemelementer og evt. forandring i eksisterende kode (skriv kun tilleggene/forandringen) for å underbygge utredningen.

Vedlegg til oppgave 3

Klasser og Interface:

- Main – Trenger ikke være med i oppgave 3.1 UML
- Person
- Eier
- Kontakt
- Ansatt
- Installasjon
    - Alarmpunkt
- Alarm
- Utrykning
- Kontaktbar

Java-dokumentasjon:

- GregorianCalendar

Vedlegget er på 8 sider inklusive denne siden.

Main

```
public class Main {

	public static void main(String[] args) {
		// registrerer ansatte
		Ansatt petter = new Ansatt("Per", "Badberg", "Holmen 3", "1792",
			"Tistedal", "91231800", "vaktleder");
		Ansatt trine = new Ansatt("Trine", "Överaskende Stärk", "Dumpa 31",
			"1758", "Halden", "47241907", "vekter");
		Ansatt tom = new Ansatt("Tom", "Heinatt", "Plassen 4", "1765",
			"Halden", "92216859", "vekter");

		// eier av en installasjon -> kunde
		Eier hans = new Eier("Hans", "Olsen", "Gata 1", "1776", "Halden",
			"9161688", "K012190");

		// kontaktpersoner til installasjonen
		Kontakt per = new Kontakt("Per", "Hansen", "Parkvn. 8", "1768",
			"Halden", "48052382");
		Kontakt kari = new Kontakt("Kari", "Pettersen", "Parkvn. 8", "1768",
			"Halden", "91022172");

		// installasjonen til Hans
		Installasjon hansInstallasjon = new Installasjon("SN1239A_12801",
			"Parkvn. 8", "1768", "Halden", hans);

		// monterer alarmpunkt i installasjonen til Hans - kun røykdetektorer
		hansInstallasjon.leggTilAlarmpunkt("Røykdetektor", "Leilighet 1
			- Soverom 1");
		hansInstallasjon.leggTilAlarmpunkt("Røykdetektor", "Leilighet 1
			- Soverom 2");
		hansInstallasjon.leggTilAlarmpunkt("Røykdetektor", "Leilighet 2
			- Soverom 1");
		hansInstallasjon.leggTilAlarmpunkt("Røykdetektor", "Leilighet 2
			- Soverom 2");
		hansInstallasjon.leggTilAlarmpunkt("Røykdetektor", "Felles
			Trappegang");

		// registrerer kontaktpersoner for alarmsentral
		// Hans bor ikke på stedet så han settes ikke som kontaktperson
		hansInstallasjon.leggTilKontaktperson(per);
		hansInstallasjon.leggTilKontaktperson(kari);

		// ********** vi tester installasjonen i Parkveien 8 **************

		// starter test (utløser alle alarmpunkt)
		Alarm testalarm = hansInstallasjon.testAvInstallasjon();

		// alle ansatte rykker ut til installasjonen
		Utrykning u1 = new Utrykning(testalarm);
		u1.sendPersonell(petter);
		u1.sendPersonell(tom);
		u1.sendPersonell(trine);

		// underveis så henter de ut alarmrapport for å få oversikt
		System.out.println( u1.getAlarm().getAlarmrapport() );
	}
}
```

Person

```
public abstract class Person {
	private String fnavn;
	private String enavn;
	private String adr;
	private String postnr;
	private String poststed;
	private String tlf;
	private static ArrayList<Person> personer = new ArrayList<>();

	public Person(String fanvn, String enavn, String adr,
			String postnr, String poststed, String tlf) {

		this.fnavn = fanvn;
		this.enavn = enavn;
		this.adr = adr;
		this.postnr = postnr;
		this.poststed = poststed;
		this.tlf = tlf;
		personer.add(this);
	}

	// get og set for alle felt finnes
}
```

Eier

```
public class Eier extends Person implements Kontaktbar{

	private String kundenummer;

	public Eier(String fanvn, String enavn, String adr,
		String postnr, String poststed, String tlf,
		String kundenummer) {
		// oppgave 3.2a gjøres her
	}

	// get og set for alle felt finnes
}
```

Kontakt

```
public class Kontakt extends Person implements Kontaktbar{

	public Kontakt(String fanvn, String enavn, String adr,
		String postnr, String poststed, String tlf) {
		// oppgave 3.2a gjøres her
	}

	// get og set for alle felt finnes
}
```

Ansatt

```
public class Ansatt extends Person implements Kontaktbar{

	// hvilken type rolle har den ansatte vekter, vaktleder, etc
	private String rolle;

	public Ansatt(String fanvn, String enavn, String adr,
		String postnr, String poststed, String tlf, String rolle){
		// oppgave 3.2a gjøres her
	}

	// get og set for alle felt finnes
}
```

Installasjon

```
public class Installasjon {

	private String sentralID;
	private String adr;
	private String postnr;
	private String poststed;
	private Eier eier;
	private ArrayList<Kontaktbar> kontakter = new ArrayList<>();
	private ArrayList<Alarmpunkt> alarmpunkt = new ArrayList<>();
	private static ArrayList<Installasjon> installasjoner = new ArrayList<>();

	public Installasjon(String sentralID, String adr, String postnr,
		String poststed, Eier eier) {

		this.sentralID = sentralID;
		this.adr = adr;
		this.postnr = postnr;
		this.poststed = poststed;
		this.eier = eier;
		installasjoner.add(this);
	}

	public void leggTilAlarmpunkt(String type, String plassering) {
		alarmpunkt.add( new Alarmpunkt(type, plassering) );
	}

	// ***** oppgave 3.4 gjøres her *****

	// metode som lager en tekstlig rapport over de alarmpunktene som er utløst
	public String lagAlarmRapport() {
		// ***** oppgave 3.2b gjøres her *****
	}

	// metode som lar alarmselskapet teste installasjonen
	public Alarm testAvInstallasjon() {
		// ***** oppgave 3.3 gjøres her *****
	}

	// get og set finnes for alle felter

	// ***** Indre klasse for Alarmpunkt i installasjonen *****
	private class Alarmpunkt {

		// hvilken type punktet er. feks brannvarsler, magnetkontakt etc
		private String type;
		private String plassering;
		private boolean utlost = false;

		public Alarmpunkt(String type, String plassering) {
			this.type = type;
			this.plassering = plassering;
		}

		// fortsetter på neste side
```

```
            // fortsettelse fra forrige side

            // metode som utløses av hendelser knyttet til alarmpunkt
            // f.eks at en røykvarsler detekterer røykkonstentrasjon som tilsier
            at det skal alarmeres
            public void alarmpunktUtlost() {
                  utlost = true;
            }

            @Override
            public String toString() {
                  return type + ": " + plassering;
            }

            // get og set finnes for alle felter
      }
}
```

Alarm

```
public class Alarm {

      private Installasjon varsler;
      private String alarmrapport;

      public Alarm(Installasjon varsler, String alarmrapport) {
            this.varsler = varsler;
            this.alarmrapport = alarmrapport;
      }

      public String getAlarmrapport() {
            return alarmrapport;
      }

      // get og set for alle felt finnes
}
```

Utrykning

```
public class Utrykning {
	private Alarm alarm;
	private ArrayList<Ansatt> utrykningspersonell = new ArrayList<>();

	public Utrykning(Alarm alarm) {
		this.alarm = alarm;
	}

	public void sendPersonell(Ansatt a) {
		utrykningspersonell.add(a);
	}

	public Alarm getAlarm() {
		return alarm;
	}

	// get og set for alle felt finnes

}
```

Kontaktbar

```
// et interface som kun benyttes for klassifisering av
kontaktpersoner
public interface Kontaktbar {

}
```

GregorianCalendar

Constructors

Constructor and Description

`GregorianCalendar()`

Constructs a default `GregorianCalendar` using the current time in the default time zone with the default `FORMAT` locale.

`GregorianCalendar(int year, int month, int dayOfMonth)`

Constructs a `GregorianCalendar` with the given date set in the default time zone with the default locale.

`GregorianCalendar(int year, int month, int dayOfMonth, int hourOfDay, int minute)`

Constructs a `GregorianCalendar` with the given date and time set for the default time zone with the default locale.

`GregorianCalendar(int year, int month, int dayOfMonth, int hourOfDay, int minute, int second)`

Constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale.

getTime

`public final Date getTime()`
Returns a `Date` object representing this `Calendar`'s time value (millisecond offset from the Epoch").

Returns:

`a Date representing the time value.`

```
public static void main(String[] args) {

      // eksempel på bruk av GregorianCalendar

      // instansierer GregorianCalendar-objekt som henter tid og dato
         «NÅ»
      GregorianCalendar start = new GregorianCalendar();

      // skriver ut dato og tid for instansiering
      System.out.println(start.getTime());

}
```