Høgskolen i Østfold

EKSAMEN

Emnekode:	**Emnenavn:**
ITF10611	Objektorientert programmering
Dato:	**Eksamenstid:**
31.mai 2016	4 timer
Hjelpemidler:	**Faglærer:**
To A4-ark (fire sider) med egne notater	Per Bisseberg

Om eksamensoppgaven og poengberegning:

Oppgavesettet består av 16 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.

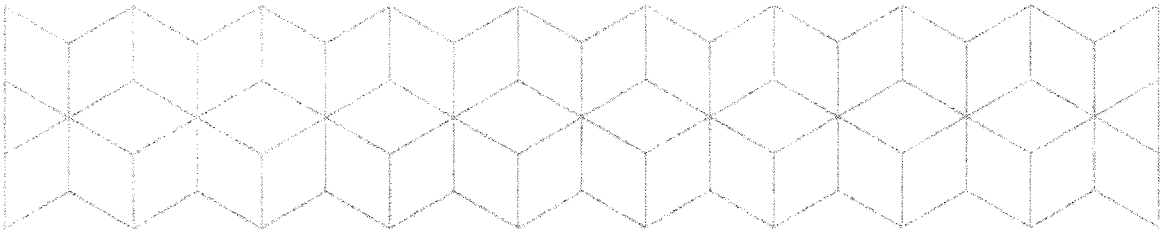
Pass også på å svare på alle oppgaver. Det er bedre å skrive litt i grove trekk hvordan du ser for deg at oppgaven kan løses, enn å ikke skrive noe i det hele tatt, dersom du står fast.

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

Lykke til!

Sensurfrist: 22. juni 2016

Karakterene er tilgjengelige for studenter på Studentweb senest 2 virkedager etter oppgitt sensurfrist.



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 polymorfi/polymorfisme og eksemplifiser med enkel kode.
b) Forklar begrepet interface/grensesnitt og eksemplifiser med enkel kode

Oppgave 1.2

Forklar disse nøkkelordene fra Java

a) final
b) static
c) override

Oppgave 1.3

Gjør rede for begrepene aggregering og komposisjon. Underbygg utredningen ved å lage et enkelt UML-diagram og eksemplifiser med kode.

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 class A {
	private String kode;;

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

public class B extends A{
	private int verdi;

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

Oppgave 2.1.2

```
public interface  A {
	void printTekst();
}

public class B extends A{
	private String navn;
	private String tekst;

	public B(String tekst, String navn) {
		this.tekst = tekst;
		this.navn = navn;
	}

	@Override
	public void printTekst() {
		System.out.println(navn + ": " + tekst);
	}
}
```

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;

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

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

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

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

	Bil volvo = new Bil("Volvo", "XC90");
	biler.add(volvo);

	Bil volvo2 = volvo;
	biler.add(volvo2);

	volvo2.setModell("XC60");

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

Oppgave 2.2.2

Vi benytter Bil-klassen fra forrige oppgave

```
public static void main(String[] args) {
	ArrayList<Bil> biler = new ArrayList<Bil>();
	biler.add(new Bil("Volvo", "XC90"));
	biler.add(new Bil("Ford", "Mondeo"));
	biler.add(new Bil("BMW", "X5"));

	Collections.sort(biler,new Comparator<Bil>() {

		@Override
		public int compare(Bil b1, Bil b2) {
			int b1L = b1.getMerke().length() + b1.getModell().length();
			int b2L = b2.getMerke().length() + b2.getModell().length();
			return b2L - b1L;
		}
	});

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

Oppgave 2.2.3

```
public static class Bil {
	private String merke;
	private String modell;
	private Kunde solgtTil;

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

	@Override
	public String toString() {
		if(solgtTil != null){
			return merke + " " + modell + ", Solgt til: " + solgtTil;
		}
		else{
			return merke + " " + modell;
		}
	}

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

public static class Kunde{
	private String fNavn;
	private String eNavn;

	public Kunde(String fNavn, String eNavn) {
		this.fNavn = fNavn;
		this.eNavn = eNavn;
	}

	@Override
	public String toString() {
		return fNavn + " " + eNavn;
	}

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

fortsetter på neste side.

```
// main metode
public static void main(String[] args) {
	ArrayList<Bil> biler = new ArrayList<Bil>();
	biler.add(new Bil("Volvo", "XC90"));
	biler.add(new Bil("Ford", "Mondeo"));
	biler.add(new Bil("BMW", "X5"));
	biler.add(new Bil("Lada", "Niva"));

	Kunde per = new Kunde("Per", "Kakse");

	Iterator<Bil> bit = biler.iterator();
	while(bit.hasNext()){
		Bil b = bit.next();
		if(b.getMerke().equals("Volvo") && b.getMerke().equals("BMW")){
			bit.remove();
		}
	}

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

Oppgave 3(50%)

Vi skal utvikle en prototype for en oppdragsgiver, det private legesenteret «***Volvoat***». Vi har nettopp begynt utviklingen og vi jobber med å få på plass 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.

Volvoat trenger et system som:

- Registrerer ansatte, de har tre ulike typer ansatte.
    - Allmennpraktiserende leger, spesialister og legesekretærer.
- Registrerer pasienter
- Registrerer timebestillinger

Reglene for 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**.
- Ansatt
    - Alle ansatte registreres med et unikt ansattnummer, for- og etternavn
- Lege
    - Alle leger registreres i tillegg med en autorisasjonskode fra myndighetene.
- Sekretær
    - (Sekretar) har ingen ytterligere informasjon.
- Allmennpraktiserende
    - (AlmPrakt) har ingen ytterligere informasjon.
- Spesialist
    - Spesialister registrerer i tillegg det medisinske spesialfeltet denne tilhører.
- Pasient
    - Pasienter skal registreres med fødsels- og personnummer (vi har fått tillatelse fra Datatilsynet), fult navn, full adresse og telefonnummer
    - Vi skal hele tiden oversikt over antall registrerte pasienter
- Time
    - Alle timebestillinger skal registrere hvilken pasient som har bestilt timen. Hvilken lege som skal behandle pasienten. Dato for timen. Hvilken sekretær som har registrert bestillingen.
    - Når en pasient har møtt til en time, så skal man via systemet kunne legge til de behandlingene som utføres i løpet av timen, med obligatorisk informasjon i form av beskrivelse og pris for behandlingen. Det skal kunne være mulig å utføre flere behandlinger i løpet av en time. Behandling er en fellesbetegnelse for alle tjenester legen yter utover konsultasjon i løpet av timen, f.eks. prøver, sykemelding, medisinsk behandling etc .
    - Prisen for timen skal kunne beregnes ut ifra de behandlingene som er ytt samt en fast konsultasjonspris. Om en pasient ikke møter til timen så blir timens pris tillagt et gebyr i tillegg til den faste konsultasjonsprisen.
- Register
    - Vi har klassen Register som holder en oversikt over alle registrerte Ansatte, Pasienter og Timer.

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 klassen Main.

Oppgave 3.2

a) Skriv ferdig konstruktørene i klassene Lege og Spesialist.
b) Klassen Time definerer metoden ***beregnPris().*** Implementer logikk i denne metoden som summerer prisen for denne timen etter reglene som er beskrevet over.
c) I løpet av en time vil en pasient kunne motta alt fra ingen til mange behandlinger, lag en metode som oppretter og lagrer behandlinger for denne timen. Metoden skal hete leggTilBehandling og ta imot relevante parametre.

Oppgave 3.3

I klassen Time har vi metoden ***lagKvittering()***
Denne metoden skal returnere en tekststreng for den aktuelle timen som ser slik ut ser slik ut:

```
Pasientens navn: Per Sjuk
Behandlende lege: Doktormann, Ole
Total sum: 1145.98
```

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

Oppgave 3.4

Vi ønsker å hele tiden ha oversikt over antall unike pasienter som har hatt timer på legesenteret. Selv pasienter som vi ikke har i registeret lenger. Gjør nødvendig endring i koden der du finner det hensiktsmessig for å oppnå dette.

Oppgave 3.5

I klassen Register har vi funksjonen ***ansattrapport()*** som inneholder nødvendig funksjonalitet for å skrive ut all informasjon om alle ansatte vi har i systemet. Vi ønsker derimot å kunne sortere de ansatte før vi skriver ut informasjonen:

a) Gjør tillegg/forandring i klassen Ansatt som definerer at den naturlige sorterings-ordenen på Ansattobjekt er alfabetisk (A-Å) på etternavn.
b) Gjør så nødvendig tillegg/forandring i metoden ***ansattrapport()*** i klassen Register som garanterer at denne vil skrive ut en sortert liste over ansatte.

Oppgave 3.6

Vi har fått en forespørsel fra legene om ekstra funksjonalitet, de ønsker seg en måte å se hvor mye penger de har loppet sine pasienter for.

Lag en metode i Registrer klassen som returnerer totalsummen en gitt lege har «håvet inn».

Vedlegg til oppgave 3

Klasser:

- Main – Trenger ikke være med i oppgave 3.1 UML
- Ansatt
- Lege
- Sekretar
- Spesialist
- AlmPrakt
- Pasient
- Time
- Register

Vedlegget er på 6 sider inklusive denne siden.

Main

```
public class Main {

	public static void main(String[] args) {
		// Ole begynner å arbeide ved legesenteret
		AlmPrakt a = new AlmPrakt(1234, "Ole", "Doktormann", "AT1234QXY125");

		// Det gjør Hans også
		Sekretar s = new Sekretar(1236, "Hans", "Onatopp");

		// Per er pjusk
		Pasient p = new Pasient("25037634158", "Per Sjuk",
			"Gata 1, 1792 Tistedal", "91262688");

		// Per bestiller en time av Hans for å bli undersøkt av Ole
		// den 31 mai 2016.
		Time t = new Time(p, a, s,
			new GregorianCalendar(2016,4,31));

		// Per melder sin ankomst til legesekretæren
		t.setMott(true);

		// Per må ta en blodprøve samt en urinprøve -> 2 behandlinger
		// NB: Ole har kjøpt seg ny Volvo XC90
		t.leggTilBehandling("Blodprøve, pas. var vrang ", 499.99);
		t.leggTilBehandling("Tisse gikk bedre for pas.", 246);

		// Per går til legesekretæren Hans for å betale
		t.beregnPris();

		// Per får kvittering av Hans
		System.out.println(t.lagKvittering());

		// Per gråter.....
	}
}
```

Ansatt

```
public abstract class Ansatt {
	private int ansattNr;
	private String fornavn;
	private String etternavn;

	//konstruktør
	public Ansatt(int ansattNr, String fornavn, String etternavn) {
		this.ansattNr = ansattNr;
		this.fornavn = fornavn;
		this.etternavn = etternavn;
		Register.getAnsatte().add(this);
	}

	// tekstlig representasjon av en ansatt
	@Override
	public String toString() {
		return etternavn + ", " + fornavn;
	}

	// anta at get og set metoder for alle felter finnes..
}
```

Lege

```
public abstract class Lege extends Ansatt{
	private String autKode;

	// konstruktør
	public Lege(int ansattNr, String fornavn, String etternavn, String autKode) {
		// oppgave 3.2a
	}

	// anta at get og set metoder for alle felter finnes..
}
```

Sekretar

```
public class Sekretar extends Ansatt{

	// konstruktør
	public Sekretar(int ansattNr, String fornavn, String etternavn) {
		super(ansattNr, fornavn, etternavn);
	}
}
```

Spesialist

```
public class Spesialist extends Lege{


	private String spesFelt;

	// konstruktør
	public Spesialist(int ansattNr, String fornavn,
			String etternavn, String autKode, String spesFelt) {
		// Oppgave 3.2a
	}

	// anta at get og set metoder for alle felter finnes..
}
```

AlmPrakt

```
public class AlmPrakt extends Lege{

	//konstruktør
	public AlmPrakt(int ansattNr, String fornavn, String etternavn, String autKode){
		super(ansattNr, fornavn, etternavn, autKode);
	}
}
```

Pasient

```
public class Pasient {
	private String fpNummer;
	private String fulltNavn;
	private String fullAdr;
	private String tlfNummer;

	// konstruktør
	public Pasient(String fpNummer, String fulltNavn,
			String fullAdr, String tlfNummer) {

		this.fpNummer = fpNummer;
		this.fulltNavn = fulltNavn;
		this.fullAdr = fullAdr;
		this.tlfNummer = tlfNummer;
		Register.getPasienter().add(this);
	}

	// anta at get og set metoder for alle felter finnes..
}
```

Time

```
public class Time {
	// fast konsultasjonspris
	public static final double KONS_PRIS = 399.99;
	// gebyr for å ikke møte til time
	public static final double IM_GEBYR = 859.96;
	private Pasient pasient;
	private Lege lege;
	private Sekretar bestiller;
	private Calendar dato;
	private double pris;
	private boolean mott;
	private ArrayList<Behandling> behandlinger = new ArrayList<>();

	// konstruktør
	public Time(Pasient pasient, Lege lege, Sekretar bestiller,
			Calendar dato) {

		this.pasient = pasient;
		this.lege = lege;
		this.bestiller = bestiller;
		this.dato = dato;
		Register.getTimeRegister().add(this);
	}

	// oppgave 3.2c lag metode her

	public String lagKvittering(){
		// Oppgave 3.3 - logikk her
	}

	public void beregnPris(){
		// oppgave 3.2b - logikk her
	}

	// anta at get og set metoder for alle felter i klassen Time finnes..

	// indre klasse for Behandling
	private class Behandling {
		private String beskrivelse;
		private double pris;

		public Behandling(String beskrivelse, double pris) {
			this.beskrivelse = beskrivelse;
			this.pris = pris;
		}

		// anta at get og set metoder for alle felter i den indre
		// klassen Behandling finnes..
	}
}
```

Register

```
public class Register {
	private static ArrayList<Ansatt> ansatte = new ArrayList<>();
	private static ArrayList<Pasient> pasienter = new ArrayList<>();
	private static ArrayList<Time> timeRegister = new ArrayList<>();

	public static void ansattrapport(){

		// oppgave 3.5b

		for(Ansatt a: ansatte){
			System.out.println(a);
		}
	}

	// oppgave 3.6 - lag metode her

	// anta at alle lister har statiske get metoder....
}
```