NTNU
Det skapende universitet

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgave i TDT4100
Objektorientert programmering med Java

Faglig kontakt under eksamen: Rune Sætre

Tlf.: 452 18 103

Eksamensdato: 2013, torsdag 16. mai

Eksamenstid (fra-til): kl 09:00-13:00

Hjelpemiddelkode/Tillatte hjelpemidler: C – kalkulator og

Kun «Big Java» av Cay S. Horstmann, utgitt av Wiley, er tillatt.

Blanke lapper for å finne fram til riktig side, gul/rosa markering av ord, og kommentarer av typen «NB», «OBS», «Les Dette» etc. er ok. Håndskrevne notater er ikke tillatt, og må fjernes fra boken før eksamen starter.

Annen informasjon:

Les oppgaveteksten nøye. Finn ut hva det spørres etter i hver oppgave.

Dersom du mener at opplysninger mangler i en oppgaveformulering, gjør kort rede for de antagelser og forutsetninger som du finner nødvendig.

Målform/språk: Bokmål

Antall sider: XXX

Antall sider vedlegg: 0

Kontrollert av:

5/5-13 _____

Dato *Hallvard Trætteberg.*

Introduksjon – For hele oppgavesettet

Systemet for innlevering av selvangivelsen og publisering av beregnet skatt i Norge er Altinn. De siste årene har det vært mange problemer med systemet, spesielt fordi alle har samme tidsfrister. Vi tenker oss nå at Staten har bestemt seg for å lage et nytt, ryddig og objektorientert skatteberegningssystem (heretter kalt TaxProgram) helt fra bunnen av. De har bestemt at TaxProgram skal programmeres i Java (nå som årets nye sikkerhetshull er tettet igjen), og de har formulert en del krav til det nye systemet. Du bestemmer deg for å implementere TaxProgram på egen hånd, og finner denne (forenklede) kravspesifikasjon fra staten:

Den «personlige» skatteberegningskomponenten (TaxProgram) må:
-Inneholde opplysninger om personer, med personnummer, navn, inntekt, fradrag, formue, gjeld og skatteprosent.
-Sjekke at alle oppgitte personnummer (idnr) er gyldige (forenklet: 11 siffer).
-Dele ut nye personnr til nyfødte, og sikre at nye personnummer er gyldige, unike og konstante.
-Avlaste Altinn ved å fordele innleveringsfristen til alle skattebetalere jevnt ut over en hel uke.
-Sjekke at inntekt, fradrag, formue og gjeld ikke er negative tall.
-Beregne skatten for alle landets innbyggere etter følgende formler:

Skatt = Inntektsskatt + Formueskatt
Inntektsskatt = (inntekt – fradrag) * skatteprosent %
Formueskatt = (formue – gjeld) * 1 %
Hvis gjelden er større enn formuen blir det ingen formueskatt det året.
(NB: Tilsvarende for inntekt/fradag for å unngå negativ skatt!!)

-Lage lands- og kommunestatistikk, med totalt innbetalt skatt pr. kommune.
-Lagre konstanter kun ett sted i programmet, mtp endringer i skatteloven.
-- f.eks. 1 % formueskatt for et bestem år.
-Ta vare på alle data fra tidligere år i en Database.
-Håndtere firma (**TaxFirm**) som en spesiell type «**TaxEntity**», hvor personnummer er byttet ut med et organisasjonsnummer (idnr), men reglene ellers er like.

- Organisasjonsnummer brukes for å identifisere *juridiske personer* (enheter) i Norge, og tildeles ved registrering i Enhetsregisteret. Organisasjonsnummeret består av ni siffer og starter på tallet 8 eller 9.
- Foretaksnavnet må inneholde minst tre bokstaver fra det norske alfabetet, se foretaksnavneloven § 2-1 første ledd. Bokstavene trenger ikke å danne et ord eller å stå etter hverandre.

Del 1 – Innkapsling og abstrakte klasser (30 %)

Da er det på tide å begynne med **TaxEntity**-klassen. Den skal inneholde alt det som er felles for **TaxPerson**- og **TaxFirm**-objekter, slik som idnr, navn, skatteprosent, inntekt, fradrag og gjeld.
Det skal ikke være mulig å opprette instanser av selve **TaxEntity**-klassen.

a) Skriv ned hele klasse-skjelettet (men **uten** *metoder og imports*) for klassen **no.ntnu.eksamen.TaxEntity**.
Hvilke modifikatorer trenger klassen, og hvorfor?
Kommenter hvilken datatype du bør bruke for feltene («instance variables»).

```
package no.ntnu.eksamen;
```

```
public abstract class TaxEntity{
	final private String idnr;		// For enkel validering, og nok plass. Konstant!
	private String name;			// Ingen krav om oppdeling
	private double percent;			// Ikke nødvendigvis heltall: 3.5%
	private long formue;			// Verdi oppgis som hele kroner, kanskje >2mrd.
	private long gjeld;			// Se formue, spesielt for store firma.
	private long inntekt;			// Se formue
	private long fradrag;			// Se formue
}
```

Klassen må være *public abstract*, slik at den kan brukes som variabeltype i **YearRegister**, men samtidig slik at den ikke kan instansieres (uten å benytte sub-klassene).

(Kommentarer bør si hvorfor datatypen er valgt!)
Instansvariablene (idnr, navn, skatteprosent, formue, gjeld, inntekt, fradrag) skal være private.
(idnr må evt. være protected hvis det skal håndteres direkte av checkId() i TaxPerson og TaxFirm).
(Formue må introduseres her nå, eller senest i oppgave 1e!)

idnr må være final for å sikre at det ikke kan endres etter initialisering.

b) Lag fire hjelpemetoder `checkPositive, checkPercent, checkName og checkId. De skal returnere` *true* hvis
-et oppgitt tall er større eller lik 0.
-en oppgitt prosent er mellom 0 og 100.
-et oppgitt navn inneholder minst 3 norske bokstaver
-en oppgitt id er gyldig.
Metodene skal returnere *false* i de motsatte tilfellene.

Hvorfor må metoden **checkId** være *abstract*?
Hvilken synlighet bør disse metodene ha?

```
/** Sjekker at tallet er større enn 0.*/
private boolean checkPositive(long number){
	return (number >= 0);
}

/** Antar at prosent aldri skal være 0 eller 100 eksakt.*/
private boolean checkPercent(double percent){
	return ( percent >= 0 && percent <= 100 );
}//checkPercent

/** Sjekker at det er minst tre norske tegn i navnet. */
private boolean checkName(String name){
	int charCount = 0;
	String lc = name.toLowerCase();
	for(int i=0; i < lc.length(); i++){
		char c = lc.charAt(i);
		if ( c >= 'a' && c <= 'z' || c=='æ' || c=='ø' || c=='å' ){
			charCount++;
		}
	}
	if (charCount < 3){
		return false;
	}
	return true;
}//checkName
```

```
/** Denne må implementeres i sub-klassene.*/
protected abstract boolean checkId(String id);
```

checkId må være *abstract* siden **TaxEntity** ikke kan vite om den er en **TaxPerson** eller **TaxFirm**.

Metodene bør være *private*, siden de inneholder spesifikke regler som kun skal brukes i **TaxEntity**, men **checkId()** må være *protected* (siden den er *abstract*) slik at den kan *Override*s i sub-klassene **TaxPerson** eller **TaxFirm**.

Man kan også argumentere for at **checkName** må behandles på en lignende måte som **checkId** (hvis man vil), men det står ikke noe om det i oppgaven.

c) Implementer metodene s**etName og setPercent**.
Hva slags synlighet bør disse ha?
Hva bør gjøres hvis det oppdages ugyldige verdier?

```
//SETTERS
public void setName(String name) {
	if ( checkName(name) ){
		this.name = name;
	}else{
		throw new IllegalArgumentException("Ugyldig navn: "+name);
	}
}

public void setPercent(double percent) {
	if ( checkPercent(percent) ){
		this.percent = percent;
	}else{
		throw new IllegalArgumentException("Ugyldig skatteprosent: "+percent);
	}
}

Setters bør være public slik at de kan benyttes fra (potensielt andre) TaxProgram etc.
```
(Bonus for riktig bruk av «this».)
Bør kaste **IllegalArgumentException** (eller annen **RuntimeException)** hvis feil oppdages.

d) Lag en *protected* konstruktør for **TaxEntity**. Den må ta inn idnr, navn og skatteprosent, og benytte hjelpemetodene over (fra b og c).

Hvorfor er det ønskelig at konstruktøren er *protected*?
Hvor kan en *protected* konstruktør kalles fra, og hvordan?

```
protected TaxEntity( String id, String name, double percent ){
	//Bruker enkle hjelpemetoder for å sjekke at id, name og prosent er korrekt
	if ( checkId(id) ){
		idnr = id;
	}else{
		throw new IllegalArgumentException("Ugyldig id: "+id);
	}
```

```
        setName( name );
        setPercent(percent);
}//CONSTRUCTOR
```

Vi trenger en «*protected* constructor» for å vise at ingen andre enn sub-klassene skal kunne opprette instanser av typen **TaxEntity**.
(Andre klasser i *package* **no.ntnu.eksamen** kunne også brukt *protected*-konstruktøren, om ikke klassen hadde vært *abstract*).

Alle kall til denne konstruktøren fra en sub-klasse (som TaxPerson eller TaxFirm) må skje i første linje i konstruktøren til sub-klassen. Syntaksen er:
super(id, name, percent);

e) Implementer tilgangsmetoden getTax(). Den skal returnere hvor mye TaxEntity må betale, gitt nåværende inntekter, fradrag, formue og gjeld.

```
/** For å hente ut antall kr beregnet skatt etter følgende regler
* Skatt = Inntektsskatt + Formuesskatt
* Inntektsskatt = (inntekt - avdrag) * skatteprosent %
* Formuesskatt = (formue – gjeld) * 1 %
* Hvis gjelden er større enn formuen blir det ingen formuesskatt det året.
* Rimelig å anta at det samme gjelder for fradrag større enn inntekter.
*/
public long getTax(){
        long tax = 0;
        if (formue > gjeld){
                tax += (formue-gjeld)*YearRegister.wealthTaxPercent/100;
        }
        if (inntekt > fradrag){
                tax += (inntekt-fradrag)*percent/100;
        }
        return tax;
}
```

f) Hvilke andre tilgangsmetoder bør implementeres i **TaxEntity**-klassen?
Hvilken synlighet skal de ha?

Trenger bare get-metode for idnr, siden det aldri skal kunne endres.
Trenger get- og set-metoder for navn, skatteprosent, inntekt, formue, fradrag og gjeld.
Alle disse tilgangsmetodene bør være public slik at de kan benyttes i (potensielt andre) TaxProgram.

Del 2 – Arv (10 %)

Implementer klassene for **TaxPerson** og **TaxFirm**.

```
package no.ntnu.eksamen;
public class TaxPerson extends TaxEntity {
        private static final int PNUM_LENGTH = 11;
```

```
	protected TaxPerson(String id, String name, double percent) {
		super(id, name, percent);
	}

	@Override
	protected boolean checkId(String id) {
		if ( id.trim().length() != PNUM_LENGTH ){
			return false;
		}else{
			for (int i=0; i<PNUM_LENGTH; i++){
				if( ! Character.isDigit( id.charAt(i) ) ){
					return false;
				}
			}//check all digits. return false immediately if a bad char is found
			return true;
		}
	}
}//class TaxPerson
```

```
package no.ntnu.eksamen;
public class TaxFirm extends TaxEntity {
	private static final int ORGNUM_LENGTH = 9;

	protected TaxFirm(String id, String name, double percent) {
		super(id, name, percent);
	}

	@Override
	protected boolean checkId(String id) {
		if ( id.length() == ORGNUM_LENGTH &&
				(id.charAt(0) =='8' || id.charAt(0) =='9') ){
			for (int i=0; i<ORGNUM_LENGTH; i++){
				if ( ! Character.isDigit( id.charAt(i) ) ){
					return false;
				}
			}// check all digits. return false immediately if a bad digits
			return true;
		}else{
			return false;
		}//wrong length or starting digit
	}
}//class TaxFirm
```

For at konstruktøren skal gjøre validering riktig, må en redefinere en hjelpe-valideringsmetode som kalles i konstruktøren i 1d).
Stikkord her er *extends* og *super*(navn, idnr, skatteprosent), og bonus for *@Override* og *trim()*

Del 3 – Programmering (40 %)

I denne oppgaven skal du implementere klasser og metoder for å håndtere skatteoppgjøret for et bestemt år. For alle oppgavene gjelder det at du kan *bruke* andre metoder deklarert i samme eller tidligere deloppgaver, selv om du ikke har implementert dem (riktig).

Du skal implementere en klasse **YearRegister** for å representere alle innbyggere og firma.
YearRegister skal inneholde metoder for…

- å opprette et tomt **YearRegister** for et bestemt år (3a)
- å si om en **TaxEntity** med idnr finnes i listen (3c)
- å finne alle personer i en bestemt Kommune (3c og 4)
- å beregne skatten for alle **TaxEntity** i **YearRegister**
(4b – **NB:** Oppgaven finnes ikke! Ble flyttet til 1e)
- å beregne maksimum, minimum og gjennomsnittlig skatt for året (3d)
- å returnere en *String* med info om år, antall, høyeste, laveste og gjennomsnittlig skatt (3a og 3d)
f.eks. `"YearRegister 2009: 4321000 TaxEntities:`
`Max tax is 100200300, Min tax is 0, and Average tax is 111000"`

a) For å lage historiske oversikter er vi nødt til å skille skattelistene for forskjellige år.
Begynn med å implementere klassen YearRegister og følgende to metoder for å **initialisere et nytt YearRegister for et bestemt år**, og for å **returnere info om «år, antall, høyeste, laveste og gjennomsnittlig skatt».**

A1 - YearRegister(int year): Initialiser et tomt YearRegister-objekt for et bestemt år**.**
A2 - String toString(): Returnere en tekstlig beskrivelse med årstall, antall personer og statistikk (se oppgave 3d).

Vi velger å bruke en Collection til å holde på alle TaxEntity.
Ved å benytte HashSet slipper man å sjekke for duplisering av TaxEntity (gitt comparable)
NB:
- Man må bruke String eller long for å få plass til pnr/orgnr.
- Man må ikke instansiere selve grensesnittet (Collection / Set / List / Map).
- public vs. private

```
public class YearRegister{

	private int year;
	private long maxTax, minTax, averageTax;
	private Collection<TaxEntity> taxList = new HashSet<TaxEntity>();

	public YearRegister(int year){
		this.year = year;
	}

	public String toString(){
		String out = "YearRegister "+year+": "+taxList.size()+" skatte-personer:\n";
		out += "Max tax is "+maxTax+", ";
		out += "Min tax is "+minTax+", ";
		out += "Average tax is "+averageTax+", ";
		return out;
	}
}//YearRegister

//maxTax, minTax, averageTax må defineres her nå, eller senest i oppgave d etterpå
```

b) Hva kalles den første metoden over (A1) og hva er spesielt med den?
Hva er spesielt med den andre metoden over (A2) og når kalles den?

A1: konstruktør – brukes for å initialisere et nyopprettet (*new)* objekt med gyldige verdier.

A2: toString() – returnerer en tekstlig representasjon av objektet, og kalles ved automatisk konvertering av objekt til String, f.eks. når println eller operatoren + brukes på String-objekter.

c) Implementer følgende metoder inkl. nødvendige deklarasjoner av felt og hjelpemetoder:
- **boolean containsTaxEntity (String idnr)**: returnerer *true* om **TaxEntity** med idnr finnes i denne lista.
- **boolean addTaxEntity (TaxEntity p)**: Legger **p** til denne lista, dersom **p** ikke finnes i lista fra før. Returverdien skal være *true* bare hvis lista faktisk ble endret.
- **boolean removeTaxEntity (TaxEntity p)**: Fjerner **p** fra denne lista. Returverdien skal være *true* bare hvis ordlista faktisk ble endret.
- **Collection<TaxEntity> getTaxEntitiesInCommune (String name)**: returnerer en samling av alle TaxEntity i denne lista som bor i kommunen **name**.

add- og remove-metodene kan brukes direkte (delegering) siden de returner true om lista ble endret. Skal en gjøre det selv, må en initialisere en lokal variabel til et uttrykk/løkke med contains og returnere denne etter add/remove-kallet.

```
public boolean containsTaxEntity(String nr) {
	for (TaxEntity p: taxList){
		if ( p.getIdnr().equals(nr) ){
			return true;
		}
	}
	return false;
}

public boolean addTaxEntity(TaxEntity p) {
	return taxList.add(p);
}

public boolean removeTaxEntity(TaxEntity p) {
	return taxList.remove(p);
}

public Collection<TaxEntity> getTaxEntitiesInCommune (String com) {
	Collection<TaxEntity> matchingEntities = new HashSet<TaxEntity>();
	for (TaxEntity p : taxList) {
		//getCommune is implemented in "Del 4".
		if (getCommune( p.getId() ).equalsIgnoreCase(com)) {
			matchingEntities.add(p);
		}
	}
	return matchingEntities;
}
```

d) Implementer metoden makeStatistics(). Den skal regne ut maxTax, minTax og averageTax for alle **TaxEntity** på lista, og lagre verdiene slik at de kan hentes ut senere med toString()-metoden (se oppgave a).
Hvordan kan vi sikre at maxTax, minTax og averageTax blir oppdatert før de brukes i toString()-metoden?
Hvilke følger får det for effektiviteten til programmet vårt?

```
// Antar at maxTax og minTax begge er satt til -1 i konstruktøren
public void makeStatistics(){
	long sumTax = 0;
	for (TaxEntity p: taxList){
		sumTax += p.getTax();
		if ( minTax == -1 || p.getTax() < minTax ){
			minTax = p.getTax();
		}
		if ( p.getTax() > maxTax ){
			maxTax = p.getTax();
		}
	}
	if (taxList.size() > 0){
		averageTax = sumTax / taxList.size();
	}
}
```

Hvis man ikke prøver å holde maxTax, minTax og averageTax kontinuerlig oppdatert i add- og remove-metodene så må man sørge for at makeStatistics()-metoden utføres aller først i toString()-metoden. Det vil i såfall føre til veldig mange (millioner) beregninger hver gang toString kalles.

Del 4 – Grensesnitt og delegering (10 %)

Posten i Norge har et register over alle **TaxEntity** sine adresser for flere år tilbake.
Registeret har følgende grensesnitt:

public **interface no.posten.IAdresse** {
	public **String** getAdresse (**String** idnr);
	public **String** getKommune (**String** adr);
	public **String** getFylke(**String** adr);
}

Klassen **no.posten.Adresseregister** implementerer grensesnittet over, og har følgende konstruktør.
public Adresseregister(int year);
Adresseregister gir altså tilgang til sist kjente adresse for en **TaxEntity** for det oppgitte året.

Forklar med kode og kommentarer hvordan du vil bruke **Adresseregister** i **YearRegister**-klassen til å implementere metoden getCommune(String idnr). Den skal returnere kommune-navnet for adressen til **TaxEntity** med idnr.

```
//Man må opprette en referanse til postens Adresseregister
private static no.posten.IAdresse posten;

Den må initialiseres med riktig år i konstruktøren, og kan deretter brukes i YearReg.
posten = new Adresseregister(year);

//For å hente ut kommunenavnet må man først slå opp idnr, og deretter adressen.
public static String getCommune(String idnr) {
	return posten.getKommune( posten.getAdresse (idnr) );
}
```

Del 5 – Modellering og Dokumentasjon (10 %)

a) For å hjelpe de som skal vedlikeholde programmet ditt må du tegne et UML klassediagram som viser forholdet mellom klassene du har brukt. Ta med minst **TaxProgram**, **YearRegister**, **TaxEntity**, **TaxPerson**, **TaxFirm** og **Adresseregister**-klassene, pluss **IAdresse**-grensesnittet**.**

TaxProgram trenger en (aggregerings-) referanse til **YearRegister**.
YearRegister trenger en aggregerings-referanse til **TaxEntity**, og referanse til **Adresseregister**.
TaxFirm og **TaxPerson** må begge arve fra **TaxEntity**.
Adresseregister implementerer «interface» **IAdresse.**

Vi er ikke så strenge på syntaks her, men se Big Java Appendix K for hvordan det bør se ut.
Bonus for å vise noen viktige attributter og metoder.

b) Hvilken metode må du ha i hovedprogramklassen din for at programmet skal kunne startes?

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

Alternativt for acm: `public void run()`