NTNU
Norges teknisk-
naturvitenskapelige universitet

Fakultet for informasjonsteknologi,
matematikk og elektroteknikk
Institutt for datateknikk
og informasjonsvitenskap

BOKMÅL

KONTINUASJONSEKSAMEN I FAG
TDT4100 Objektorientert programmering

Onsdag 6. aug 2008
Kl. 09.00 – 13.00

Faglig kontakt under eksamen:
Hallvard Trætteberg, tlf (735)93443 / 918 97263

Tillatte hjelpemidler:

- Én og kun én trykt Java-lærebok, f.eks. Liang: Introduction to Java Programming.

Sensuren faller 3 uker etter eksamen og gjøres deretter kjent på http://studweb.ntnu.no/.

Prosentsatser viser hvor mye hver oppgave teller innen settet.

Merk: All programmering skal foregå i Java.

Lykke til!

Rammen rundt oppgaven er mobildingser, av ulike merker, modeller og med ulike egenskaper. For hver oppgave vil vi først beskrive en del begreper og sammenhengen mellom dem, og så beskrive i mer detalj det du skal implementere. *Dersom du ikke klarer å implementere en eller flere metoder, så prøv å forenkle dem så mye som du trenger for å få det til. Husk at du likevel kan bruke dem i andre oppgaver, slik at du viser hva du kan!* Innfør gjerne hjelpemetoder for å gjøre løsningen ryddigere.

For alle metoder du implementerer så skal du selv velge fornuftige modifikatorer (synlighet og evt. static), datatyper og navn, dersom det ikke er oppgitt. Pass også på å sjekke om parametre er gyldige og bruke fornuftige unntakstyper for å si fra om feil.

OPPGAVE 1 (10%): Klasse for Land - Country

Alle mobildingser er knyttet til et land. I virkeligheten skjer dette gjennom abonnementet/SIM-kortet, men her forenkler vi det. Hvert land identifiseres med en to-bokstavskode og har i tillegg en landskode på to siffer, 01-99, som må brukes når en ringer til et annet land enn mobildingsen er knyttet til. Knytningen til land kan endres, tilsvarende det som skjer i virkeligheten ved å bytte abonnement/SIM-kort.

a) Vi antar at settet med land kun omfatter Danmark, Sverige og Norge. Implementer en enum-klasse kalt **Country**, med verdiene **DK**, **SE** og **NO** og landskodene 45, 46 og 47, hhv. Implementer metodene **getCountryCode**, som returnerer landskoden for landet.

```
public enum Country {

	DK(45), SE(46), NO(47);

	private int countryCode;

	private Country(int countryCode) {
		this.countryCode = countryCode;
	}

	public int getCountryCode() {
		return countryCode;
	}
}
```

b) Implementer metoden **getCountryForCode**, som returnerer **Country**-objektet for en gitt kode (eneste parameter). F:eks. med 47 som parameter skal den returnere **Country**-objektet for Norge.

```
	public static Country getCountryForCode(int countryCode) {
		for (Country country: Country.values()) {
			if (country.countryCode == countryCode) {
				return country;
			}
		}
		return null;
	}
```

OPPGAVE 2 (30%): Klasse for mobildings - MobileThing

Alle mobildingser har et merke, f.eks. ”Nokia” eller ”Sony Ericsson”, og en modellbetegnelse, f.eks. ”E60”, som sammen *identifiserer* typen mobil.

a) Implementer felt, konstruktør og tilgangsmetoder for egenskapene **brand** (merke), **model** (modell) og **country** (land), basert på opplysningene i denne og forrige oppgave. Legg vekt på å bruke fornuftige typer, modifikatorer, navn på både felt og metoder og parametre for konstruktør(er) og metoder.

```
	private final String brand, model;

	private Country country;

	public MobileThing(String brand, String model) {
		this.brand = brand;
		this.model = model;
	}

	public String getBrand() {
		return brand;
	}

	public String getModel() {
		return model;
	}
```

```
	public void setCountry(Country country) {
		this.country = country;
	}

	public Country getCountry() {
		return country;
	}
```

b) Anta at følgende statiske metode finnes i klassen SMS:

```
public static boolean sendSMS(Country country, String number, String text) { … }
```

Denne sender **text** med maksimum 128 tegn til mobilnummeret **number** i landet **country**. Ingen parametre kan være **null**. Metoden returnerer **true** dersom alt går greit, og **false** dersom noe går galt.

Implementer en metode **sendMessage** i **MobileThing**, som tar inn et nummer og tekst (uten lengdebegrensning), splitter den opp i deler på maksimum 128 tegn og sender det som *én eller flere SMS'er* vha. **SMS**-klassen sin **sendSMS**-metode. En evt. landskode gis inn som en del nummeret, med + foran (f.eks. "+4791897263"). Dersom landskode ikke er oppgitt skal SMS'en sendes til samme land som mobildingsen er tilknyttet. Metoden skal returnere antall SMS'er som meldingen ble splittet opp i og som ble sendt. Evt. feil skal videreformidles fra din metode som et 'unchecked' unntak.

```
	public int sendMessage(String number, String text) {
		Country country = this.country;
		if (number.startsWith("+")) {
			country =
Country.getCountryForCode(Integer.parseInt(number.substring(1, 3)));
			number = number.substring(3);
		}
		int count = 1;
		while (text.length() > 128) {
			String sms = text.substring(0, 128);
			text = text.substring(128);
			if (! SMS.sendSMS(country, number, sms)) {
				throw new RuntimeException("Could not send SMS to " +
number);
			}
			count++;
		}
		if (! SMS.sendSMS(country, number, text)) {
			throw new RuntimeException("Could not send SMS to " + number);
		}
		return count;
	}
```

c) sendSMS-metoden bruker returverdien for å angi at noe gikk galt. Foreslå en alternativ måte å håndtere feil på i **SMS** sin **sendSMS**-metode, som er mer i tråd med Java-måten å gjøre det på. Vis hvordan din **sendMessage**-metode evt. må skrives om, slik at den fortsatt kun avslutter ved å returnere normal eller vha. et 'unchecked' unntak.

```
	public static void sendSMS(Country country, String number, String text)
throws Exception { … }

	public int sendMessage(String number, String text) {
		…
		try {
			int count = 1;
```

```
                    while (text.length() > 128) {
                          String sms = text.substring(0, 128);
                          text = text.substring(128);
                          SMS.sendSMS(country, number, sms);
                          count++;
                    }
                    SMS.sendSMS(country, number, text);
                    return count;
              } catch (Exception e) {
                    throw new RuntimeException("Could not send SMS to " + number,
e);
              }
        }
```

OPPGAVE 3 (15%): Klasser for mobiltelefon og smarttelefon – MobilePhone og SmartPhone.

I denne deloppgaven skal MobileThing gjøres abstrakt og to subklasser for ulike mobiltyper introduseres, med ulike tastatur. Alle mobildingser har tastatur, men hvilke taster som er tilgjengelige er avhengig av om den er en smarttelefon eller en vanlig mobiltelefon og om den er åpen (klappet opp) eller lukket (klappet igjen). En smarttelefon har alltid talltastene tilgjengelig, og har ekstra qwerty-tastatur tilgjengelige i åpen tilstand. En mobiltelefon, derimot har ingen taster i lukket tilstand, og har talltaster når den er åpen (klappet opp).

a) Hva er en abstrakt metode og hva er sammenhengen mellom abstrakte metoder og klasser? Definer en abstrakt metode **getAvailableKeys** i **MobileThing**, som returnerer alle tilgjengelige taster som en **String**. F.eks. har en smarttelefon tastene ”0...9” tilgjengelig når den er lukket og ”0...9A...Åa...å” når den er åpen.

En abstrakt metode er en metode som deklareres (som abstrakt), men ikke inneholder implementasjonskode. En klasse som inneholder en abstrakt metode må selv være abstrakt (men ikke nødvendigvis omvendt). En abstrakt metode må implementeres i en ikke-abstrakt subklasse.

```
public abstract String getAvailableKeys();
```

b) Anta det finnes en metode **isKeyboardOpen**, som returnerer åpnet/lukket -statusen til mobildingsen. Definer først to **String**-*konstanter* **NUMBER_KEYS** og **QWERTY_KEYS** for hhv. alle tall- og alle bokstavtaster..Implementer deretter **getAvailableKeys**-metoden i subklassene **MobilePhone** og **SmartPhone**, basert på åpnet/lukket-statusen til mobildingsen iht. beskrivelsen over, hvor du bruker disse konstantene.

I MobileThing:

```
      protected final static String NUMBER_KEYS = "0123456789";
      protected final static String QWERTY_KEYS = "QWERTYUIOPÅASDFGHJKLØÆZXCVBNM";
```

I MobilePhone:

```
      public String getAvailableKeys() {
            return (isKeyboardOpen() ? NUMBER_KEYS : "");
      }
```

I MobilePhone:

```
	public String getAvailableKeys() {
		return (isKeyboardOpen() ? NUMBER_KEYS + QWERTY_KEYS : NUMBER_KEYS);
	}
```

c) Implementer metoden **isKeyAvailable**, som tar inn en **char** og returnerer om denne tasten er tilgjengelig.

I MobileThing:

```
	public boolean isKeyAvailable(char key) {
		return getAvailableKeys().indexOf(key) >= 0;
	}
```

OPPGAVE 4 (30%): Klasse for prisliste - PriceList

En prisliste for mobildingser har oversikt over alle mobiltypene i markedet og deres pris og kan sortere mobiler på minnestørrelse eller prisen. I denne oppgaven skiller vi mellom å legge mobildingser til lista (**MobileThing**- instanser) og det å knytte prisinformasjon til mobildingsene. Merk at samme mobildings kan ha forskjellig pris i ulike lister (tenk deg f.eks. at Lefdal og Elkjøp har hver sine lister).

a) Implementer felt og metoder for å legge til og fjerne mobildingser fra prislista.

```
	private List<MobileThing> mobiles = new ArrayList<MobileThing>();

	public void addMobileThing(MobileThing mobileThing) {
		mobiles.add(0, mobileThing);
	}

	public void removeMobileThing(MobileThing mobileThing) {
		mobiles.remove(mobileThing);
	}
```

b) Implementer en metode **getMobilesForBrand** som tar inn brand (merke) og som returnerer en liste av alle mobildingsene i prislista som er av dette merket.

```
	public List<MobileThing> getMobilesForBrand(String brand) {
		List<MobileThing> mobiles = new ArrayList<MobileThing>();
		for (MobileThing mobileThing: mobiles) {
			if (brand.equals(mobileThing.getBrand())) {
				mobiles.add(mobileThing);
			}
		}
		return mobiles;
	}
```

c) Implementer felt og metoder for å sette og returnere pris for mobildingsene. En skal ikke kunne sette prisen til en mobildings som ikke finnes i lista. Metoden som returnerer prisen til en mobildings skal returnere -1 dersom ingen pris er registrert i lista.

```
	private Map<MobileThing, Integer> prices = new HashMap<MobileThing,
Integer>();

	public void setMobileThingPrice(MobileThing mobileThing, int price) {
		if (! mobiles.contains(mobileThing)) {
```

```
                    throw new IllegalArgumentException("MobileThing is not in this
PriceList: " + mobileThing);
                }
                prices.put(mobileThing, price);
        }

        public int getMobileThingPrice(MobileThing mobileThing) {
                Integer price = (mobileThing != null ? prices.get(mobileThing) :
null);
                return price != null ? price : -1;
        }
```

d) Anta at **MobileThing**-klassen har metoden **getMemory**, som returnerer mengden minne som mobildingsen har. Implementer metodene **sortOnMemory** og **sortOnPrice**, som begge tar inn en liste med mobildingser og som sorterer lista på hhv. minne og pris. Utnytt grensesnittene **Comparable** og **Comparator** og **Collections**-klassene sine **sort**-metoder og definer nødvendige metoder i **MobileThing**- og **PriceList**-klassene.

I MobileThing:

class MobileThing implements Comparable<MobileThing>

```
        public int compareTo(MobileThing mt) {
                return initialMemory - mt.initialMemory;
    }
```

I PriceList:

class PriceList **implements** Comparator<MobileThing>

```
        public void sortOnMemory(List<MobileThing> mobiles) {
                Collections.sort(mobiles);
        }

        private int getMobileThingPrice(MobileThing mobileThing) {
                Integer price = (mobileThing != null ? prices.get(mobileThing) :
null);
                return price != null ? price : -1;
        }

        public int compare(MobileThing mt1, MobileThing mt2) {
                int p1 = getMobileThingPrice(mt1);
                int p2 = getMobileThingPrice(mt2);
                return p1 - p2;
        }

        public void sortOnPrice(List<MobileThing> mobiles) {
                Collections.sort(mobiles, this);
        }
```

OPPGAVE 5 (15%): Observatør-observert-teknikken

Mange mobilbrukere er interessert i å få SMS med informasjon om endringer i mobilmarkedet, inkl. nye modeller og endrede priser. Forklar med tekst og kodesnutter hvordan du kan gjøre **PriceList**-klassen *observerbar*, for å støtte realisering av en slik SMS-tjeneste. Vær konkret mht. navn og virkemåte for klasser/grensesnitt og metoder du velger å introdusere.

I PriceListListener:

```
public interface PriceListListener {

	public void mobileThingAdded(PriceList priceList, MobileThing mobileThing);
	public void mobileThingRemoved(PriceList priceList, MobileThing mobileThing);

	public void priceChanged(PriceList priceList, MobileThing mobileThing, int
newPrice);
}
```

I MobileThing:

```
	private List<PriceListListener> priceListListeners = new
ArrayList<PriceListListener>();

	public void addPriceListListener(PriceListListener priceListListener) {
		priceListListeners.add(priceListListener);
	}

	public void removePriceListListener(PriceListListener priceListListener) {
		priceListListeners.remove(priceListListener);
	}

	public void addMobileThing(MobileThing mobileThing) {
		if (! mobiles.contains(mobileThing)) {
			mobiles.add(0, mobileThing);
			for (Iterator<PriceListListener> iterator =
priceListListeners.iterator(); iterator.hasNext();) {
				iterator.next().mobileThingAdded(this, mobileThing);
			}
		}
	}

	public void removeMobileThing(MobileThing mobileThing) {
		if (mobiles.contains(mobileThing)) {
			mobiles.remove(mobileThing);
			for (Iterator<PriceListListener> iterator =
priceListListeners.iterator(); iterator.hasNext();) {
				iterator.next().mobileThingRemoved(this, mobileThing);
			}
		}
	}

	public void setMobileThingPrice(MobileThing mobileThing, int price) {
		if (! mobiles.contains(mobileThing)) {
			throw new IllegalArgumentException("MobileThing is not in this
PriceList: " + mobileThing);
		}
		prices.put(mobileThing, price);
		for (Iterator<PriceListListener> iterator =
priceListListeners.iterator(); iterator.hasNext();) {
			iterator.next().priceChanged(this, mobileThing, price);
		}
	}
```