NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

BOKMÅL

EKSAMEN I FAG
TDT4100 Objekt-orientert programmering

Fredag 3. juni 2005
KL. 09.00 – 13.00

Faglig kontakt under eksamen:
Hallvard Trætteberg, tlf (735)93443 / 918 97263
Trond Aalberg, tlf (735)97952 / 976 31088

Tillatte hjelpemidler:

- Lewis & Loftus: Java Software Solutions (alle utgaver)
- Winder & Roberts: Developing Java Software
- Lervik & Havdal: Programmering i Java
- Mughal, Hamre & Rasmussen: Java som første programmeringsspråk
- Arnold, Gosling & Holms: The Java Programming Language
- Lervik & Havdal: Java the UML Way
- Liang: Introduction to Java programming
- Lewis & Loftus: Java Software Solutions
- Horton: Beginning Java 2 SDK 1.4 Edition
- Brunland,Lingjærde & Maus: Rett på Java
- Lemay/Cadenhead: SAMS Teach Yourself Java 2 platform in 21 days.

Sensurdato:
24. juni 2005. Resultater gjøres kjent på http://studweb.ntnu.no/ og sensurtelefon 81 54 80 14.

Prosentsatser viser hvor mye hver oppgave teller innen settet.

Merk: All programmering skal foregå i Java.

Lykke til!

OPPGAVE 1 (10%): Iterasjon.

a) Skriv kode for en metode `Object[] forskyv1(Object[] tabell)` som returnerer en kopi av `tabell` med alle elementene forskjøvet ett hakk oppover. I resultat-tabellen skal altså element n+1 være lik element n i `tabell`, mens det siste elementet i tabell skal ligge først i resultat-tabellen.

b) Skriv kode for en metode `List forskyv2(Iterator it)` som returnerer en `ArrayList` med alle elementene som `it` "genererer" (med `next()`). Elementene skal være forskjøvet ett hakk, slik at første element som `it.next()` returnerer skal være element nummer to i resultatlista, mens det siste elementet som `it.next()` returnerer skal ligge først i resultatlista. Metodene i `Iterator`-grensesnittet er som følger.

```
// Returns true if the iteration has more elements.
boolean hasNext()

// Returns the next element in the iteration.
Object next()
```

Nødvendige metoder i `List`-grensesnittet (som `ArrayList` implementerer):

```
// Appends the specified element to the end of this list.
boolean add(Object o)

// Inserts the specified element at the specified position.
void  add(int index, Object element)

// Returns the element at the specified position in this list.
Object get(int index)

// Replaces the element at the specified position in this list
// with the specified element.
Object set(int index, Object element)

// Returns the number of elements in this list.
int size()
```

OPPGAVE 2 (10%): Klasser og arv.

Gitt følgende klassedefinisjoner:

```
public class Baseklasse {
   public int i;
   public Baseklasse(int i) {
      this.i = i;
   }
   public String toString(int i) {
      return "[" + (this.i + i) + "]";
   }
   public String toString() {
      return toString(i);
   }
}
```

```
public class Subklasse extends Baseklasse {
   public Subklasse() {
      super(4);
   }
   public String toString() {
      return toString(i + i);
   }
}
```

a) Hva skrives ut når følgende kode kjøres:

```
    System.out.println(new Baseklasse(4));
    System.out.println(new Subklasse());
```

b) Hva skrives ut dersom du fjerner parentesene rundt `this.i + i` i `Baseklasse` sin `String toString(int i)`-metode? Forklar hvorfor.

c) Det er ikke bra å ha attributter definert med `public`, de bør heller innkapsles med tilgangsmetoder. Skriv nye versjoner av `Baseklasse` og `Subklasse`, hvor `i`-attributtet er innkapslet.

OPPGAVE 3 (45%): Klasser, grensesnitt og arv.

I denne oppgaven skal vi jobbe med grensesnitt for og implementasjon av klasser for lister, analogt med klassen java.util.List og java.util.ArrayList.

a) I Java bruker vi nøkkelordet ”interface” for å definere såkalte grensesnitt, i motsetning til vanlige klasser, hvor ”class” brukes.
   - Hva kan en ikke ha med i en grensesnittdefinisjon som en kan ha med i en klassedefinisjon og hvorfor?

   Navn på klasser og grensesnitt kan opptre mange steder i javakode, men det finnes noen få begrensninger på hvor hver av disse kan opptre:
   - Når kan navn på grensesnitt brukes men ikke navn på vanlige klasser?
   - Når kan navn på vanlige klasser brukes men ikke navn på grensesnitt?

b) Gitt følgende grensesnittdefinisjon:

```
public interface Liste {
   // size() returnerer antall elementer i lista
   public int size();

   // Returnerer element nr. indeks i lista.
   // Merk at 0 er første element.
   public Object get(int indeks);

   // Endrer element nr. indeks i lista til o.
   // Merk at 0 er første element.
   public void set(int indeks, Object o);
}
```

- Skriv kode for klassen ListeImpl som implementerer dette grensesnitt og som bruker en vanlig java-tabell (array) for å holde verdiene. Merk at lister av typen Liste i deloppgavene b) og c) ikke kan endre størrelse.
- Definér en konstruktør som tar inn en heltallsverdi som eneste parameter, og som gir java-tabellen denne størrelsen.

c) Skriv kodelinjer for å opprette et ListeImpl-objekt og sette elementene 0, 1 og 2 til tallene 0, 1 og 2.

d) Gitt følgende grensesnittdefinisjon:

```
public interface DynamiskListe ??? Liste {
   // Øker størrelsen til lista med 1 og legger objekt o inn,
   // slik at det får posisjon indeks.
   // Elementer med større eller lik indeks forskyves ett hakk opp.
   public void add(int indeks , Object o);

   // Fjerner elementet i posisjon indeks fra lista.
   // Størrelsen minker med 1, og
   // elementer med større indeks forskyves ett hakk ned.
   public void remove(int indeks);
}
```

- Hvilket nøkkelord må du erstatte "???" med for at dette skal være lovlig javakode?
- Hvilke egenskaper har objekter som er deklarert til å være av typen DynamiskListe?

- Skriv kode for en subklasse av ListeImpl ved navn DynamiskListeImpl, som implementerer DynamiskListe. Les kommentarene i grensesnittdefinisjonen nøye før du skriver koden!
- DynamiskListeImpl skal ha én konstruktør med tom parameterliste som gjør at lista blir opprettet med størrelsen 0.

e) Vi ønsker å implementere en listetype som begrenser hvilke typer objekter som kan legges i lista. Først defineres følgende grensesnitt:

```
public interface BegrensetListe {
   // Angir om det er lov å legge objektet o inn i lista.
   public boolean accepts(Object o);
}
```

- Skriv kode for en subklasse av DynamiskListeImpl kalt Nummerliste. Nummerliste skal implementere BegrensetListe, slik at det kun er lov å legge tall (både heltall og desimaltall) inn i lista. Definer en passende accepts-metode og redefiner nødvendige andre metoder fra DynamiskListeImpl, slik at det ikke er mulig å legge annet enn tall inn i lista. Legg vekt på å ikke gjenta kode som allerede finnes i DynamiskListeImpl. Utnytt den heller ved å bruke super-nøkkelordet.
- Sørg for at det "kastes" et unntak av typen IllegalArgumentException, om en prøver å legge inn andre typer objekter.

f) Det meste av koden du skrev i e) vil være nokså uavhengig av hvilke(n) type(r) objekter det er lov å legge inn i lista. Faktisk kan en skrive klassen slik at det kun er accepts-metoden som må endres, om en skal begrense lista til andre typer objekter.

- Skriv koden for en klasse AbstraktBegrensetListe, som er slik at subklasser kun trenger å implementere accepts-metoden. F.eks. skal følgende klassedefinisjon være nok for å begrense elementene til kun å være av typen String.

```
public class Stringliste extends AbstraktBegrensetListe {
   // begrenser lista til å kun akseptere String-objekter
   public boolean accepts(Object o) {
      return (o instanceof String);
   }
}
```

OPPGAVE 4 (20%): Testing

a) Definer 2 relevante regler for oppførsel for Liste-grensesnittet definert i oppgave 3. Vis hvordan en kan skrive testkode som tester at ListeImpl følger disse reglene.

b) Du skal lage testmetoder i en tenkt TestCase-subklasse med JUnit-rammeverket. Skriv to metoder for å teste DynamiskListe-implementasjonen din, en for å teste add-metoden og en for å teste remove-metoden. Bruk metodene `assertEquals(Object, Object)` og `assertTrue(boolean)` i TestCase for å sjekke relevante verdier. Merk at du kun skal bruke metoder som er definert i DynamiskListe og ikke andre metoder som du har i din DynamiskListeImpl-klasse.

c) Tilsvarende som i deloppgave b) skal du lage en testmetode for å teste Nummerliste-klassen fra oppgave 3. Testmetoden skal sjekke om Nummerliste begrenser hvilke elementer som kan legges inn på riktig måte. Du må både sjekke at accepts-metoden virker som den skal og at det kastes et unntak av typen IllegalArgumentException dersom en forsøker å legge inn objekter som ikke er tall.

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

Du ønsker å lage en klasse Listesum, som observerer hvordan elementene i Nummerliste-klassen endrer seg, slik at den hele tiden har en oppdatert sum av tallene i lista. Listesum vil altså fungere som *observatør*, mens Nummerliste vil være *observert*. Som en start, definerer du følgende lyttergrensesnitt, som Listesum må implementere:

```
public interface Listeendringslytter {
   // Kall til  listeEndret-metoden angir at endretListe er endret
   // i intervallet fra og med fra til og med til
   public void listeEndret(Liste endretListe, int fra, int til);
}
```

a) Anta at en Nummerliste kun kan ha én Listeendringslytter knyttet til seg. Hvordan må du endre Nummerliste for at den skal fungere som observert og si fra om endringene.

b) Forklar med tekst og kode hvordan Listesum-klassen må virke, for at et privat sum-attributt av typen int hele tiden skal være oppdatert iht. Nummerliste-objektet det lytter på. Tegn sekvensdiagram som viser hva som skjer når Nummerliste-objektet endres, med set-, add- og remove-metodene.