

## T E N T A M E N för Objektorienterad programvarututv IT, fk

DAG : 19 december 2007

Tid : 14.00-18.00

SAL : M

- Ansvarig : Bror Bjerner, tel 772 10 29, 55 54 40  
Resultat : Meddelas så fort som möjligt  
dock absolut senast den 10/1 -08  
Hjälpmedel : X.Jia: *Object-Oriented Software Development Using Java  
edition 1 och 2*  
Betygsgränser : **CTH** 3 : 24 p, 4 : 36 p, 5 : 48 p  
Betygsgränser : **GU** Godkänt : 24 p, Väl godkänt : 45 p

### OBSERVERA NEDANSTÅENDE PUNKTER

- Börja varje ny uppgift på nytt blad.
- Skriv personnummer på varje blad.
- Använd bara ena sidan på varje blad.
- Klumpiga, komplicerade och/eller oläsliga delar kan ge poängavdrag.
- **L y c k a t i l l ! ! !**



**Uppg 1:** Allmänna frågor. Svara med ja eller nej.

- a) Implementerar gränssnittet `SortedSet<E>` gränssnittet `Collection<E>` ? (2 p)
- b) Om A är en subclass till B, gäller då att `ArrayList<A>` är en delmängd till `ArrayList<B>` ? (2 p)
- c) Är `Collection<String>` en superklass till `ArrayList<String>` ? (2 p)
- d) Går det att komma åt en `private`-deklarerad variabel från någon annan klass, subclass eller superklass ? (2 p)
- e) Om vi gör jämförelserna `new Integer(5) == 5` respektive `new Integer(5) == new Integer(5)`. Kommer de att ge samma resultat ? (2 p)

**Uppg 2:** Skriv ett program `FindLines` som räknar ut på hur *många rader en viss siffra förekommer* i en given fil. Siffran och filen ges via kommandoraden. Du kan vidare förutsätta att programmet skall exekveras i det bibliotek som filen finns. Dvs programmet exekveras och ger utskrift enligt exempelvis:

```
> java FindLines 7 kalle.txt
Siffran 7 förekom på 12 rader.
>
```

Vid felaktiga argument på kommandoraden vid anrop skall ett felmeddelande ges, som visar hur programmet skall användas.

(12 p)

**Uppg 3:** Givet följande gränssnitt:

```
public interface Queue<E> {
    /**
     * Lägg till argumentet elem längst bak i kön
     */
    public void enqueue( E elem );

    /**
     * Tag bort första elementet i kön och
     * returnera det.
     * Kasta NoSuchElementException om kön är tom.
     */
    public E dequeue();

    /**
     * Returnera första elementet i kön.
     * Kasta NoSuchElementException om kön är tom.
     */
    public E front();

    /**
     * Returnera antalet element i kön.
     */
    public int size();
} // interface Queue
```

En `Queue<E>` är en datastruktur där nya element alltid läggs sist i kön och där det alltid är det första elementet som hämtas, precis som en vanlig kö i någon affär. Dvs. den följer FIFO-principen (First In First Out).

Din uppgift är att definiera en klass `MyQueue<E>` som implementerar gränssnittet `Queue<E>`. Implementeringen skall delegera det mesta av sitt jobb till datastrukturen `LinkedList<E>` i `util`-paketet. (se utdrag i bilaga). **(12 p)**

**Uppg 4:** Givet att vi har en mängd personer, med den enkla implementeringen

```
public class Person {
    protected String idNumber;
    protected String name;

    public Person( String idNumber, String name ) {
        this.idNumber = idNumber;
        this.name      = name;
    }

    public String getId() {
        return idNumber;
    }
    public String getName() {
        return name;
    }
}
```

Vidare har vi samlat ihop en massa personer i ett en mängd `Set<Person>`, som vi nu vill sortera efter ålder där yngst skall komma först. Variabeln `idNumber` innehåller det vanliga personnumret, varför du vet åldern via de 6 första siffrorna.

För att sortera dem, så skall du använda dig av `SortedSet<E>`. Problemet är bara att `Person` inte är jämförbar. Du skall därför göra en subclass till `Person` som är jämförbar enligt gränssnittet `Comparable<E>` och där 'minsta' person är detsamma som yngsta person. Du kan anta att ingen person är mer än 99 år gammal.

Vidare skall i denna subclass finnas en klassmetod som tar ett `Set<Person>` som argument och som resultat ger en textsträng där det finns en person per rad i åldersordning (yngst först).  
Poängfördelning:

- a) Klasshuvud och konstruktör (4 p)
- b) `compareTo`-metoden (4 p)
- c) klassmetoden (4 p)

**Uppg 5:** Givet klassen:

```
public class MyThread extends Thread {  
  
    private int whatToAdd;  
    private int howMany;  
  
    public static int shared = 0;  
  
    public MyThread( int toAdd, int times ) {  
        whatToAdd = toAdd;  
        howMany    = times;  
    } // constructor MyThread  
  
    public void run() {  
        int temp = 0;  
        for ( int i = 0; i < howMany; i++ ) {  
            temp    = shared + whatToAdd;  
            shared = temp;  
        }  
    } // run  
  
} // class MyThred
```

Varje sådan tråd adderar talet i `whatToAdd` till `shared` antalet i `howMany` gånger. (Notera att `whatToAdd` kan vara negativ.) `shared` delas av alla instanser av `MyThred`, eftersom den är deklarerad `static`

- a) Definiera en klass `IncAndDecrOneMillion`, så att `main`-metoden startar två trådar. Den första tråden skall addera +1 till `shared` en miljon gånger, den andra skall addera -1 till `shared` en miljon gånger. Efter att ha startat trådarna skall `main`-metoden vänta tills dess att båda trådarna har avslutats och därpå skriva ut värdet i `shared` (Tips: kolla `join` i Jia 11.1.2 ) **(5 p)**
- b) Varför kommer utskriften vanligtvis inte bli 0 ? **(2 p)**
- c) Modifiera klassen `MyThread` så att exekvering av `IncAndDecrOneMillion` alltid skriver ut 0. Modifieringen

skall fortfarande tillåta att trådarna exekveras 'parallellt', dvs att exekveringen än lägger till 1 och än lägger till -1.

Tips: Om du vill du kan använda dig av en synkroniserad sats (se Jia 11.2.1). För att använda samma lock för båda trådarna så kan du göra synkroniseringen över ett objekt lockObj som lämpligen deklarerats med

```
public static Object lockObj = new Object();
```

**(5 p)**

- d)** Modifiera klassen IncAndDecrOneMillion så att antalet iterationer ges av två tal på kommandoraden. Det första talet skall vara antalet gånger som +1 adderas, och det andra talet skall vara antalet gånger som -1 adderas. (Ingen extra felhantering behövs för detta.) **(2 p)**

Institutionen för  
Datavetenskap  
CTH, GU

VT07  
TDA550, DIT720  
07-12-19

**Lösningförslag till tentamen i**  
**Objektorienterad programvarutveckling IT,**  
**fk.**

DAG : 19 december 2007





- Uppg 1:**
- a) Nej, gränssnit implementerar ingenting !
  - b) Nej, generiska typer är bara subklasser till '?', dvs `ArrayList<A>` och `ArrayList<B>` är båda subklasser till `ArrayList<?>`
  - c) Ja, med samma parametertyp gäller de vanliga reglerna.
  - d) Nej, `private` betyder enbart åtkomliga i klassen
  - e) Nej, `new Integer(5) == 5` blir `true`, eftersom automatisk 'unboxing' sker, medan `new Integer(5) == new Integer(5)` ger `false`, eftersom det är två olika instanser.

```

Uppg 2: import java.io.*;
import java.util.*;

public class FindLines {

    private static void errExit(String errMessage) {
        System.err.println( errMessage );
        System.err.println(
            "Usage: java FindLines <digit> <textfile>");
        System.exit(0);
    } // printError

    public static void main( String[] args ) {
        try {
            char n = '0';
            if ( args[0].length() == 1 &&
                Character.isDigit(args[0].charAt(0)) )
                n = args[0].charAt(0);
            else
                errExit( " First argument not a digit." );
            Scanner inFile = new Scanner(new File(args[1]));
            int noRows = 0;
            while( inFile.hasNextLine() )
                if ( inFile.nextLine().indexOf(n) > -1 )
                    noRows++;
            System.out.println( "Siffran " + n + " förekom på "
                + noRows + " rader." );
            inFile.close();
        }
        catch (ArrayIndexOutOfBoundsException aioob) {
            errExit( "Missing argument(s) " );
        }
        catch (FileNotFoundException fnfe) {
            errExit( "File " + args[1] + " not found " );
        }
    } // main
} // class FindLines

```

```

Uppg 3: import java.util.*;

public class MyQueue<E> implements Queue<E>{

    private LinkedList<E> que = new LinkedList<E>();

    public void enqueue( E elem ) {
        que.addLast(elem);
    } // enqueue

    public E dequeue()
        throws NoSuchElementException {
        if ( que.size() > 0 )
            return que.removeFirst();
        else
            throw new NoSuchElementException(
                "Empty queue in dequeue");
    } // dequeue

    public E front()
        throws NoSuchElementException {
        if ( que.size() > 0 )
            return que.getFirst();
        else
            throw new NoSuchElementException(
                "Empty queue in front");
    } // front

    public int size() {
        return que.size();
    }

} // class MyQueue

```

Uppg 4: import java.util.\*;

```
public class PersonByAge
    extends Person
    implements Comparable<PersonByAge> {

    public PersonByAge( String idNumber,
                       String name      ) {
        super( idNumber, name );
    } // constructor PersonByAge

    public int compareTo( PersonByAge pba ) {
        String today      = "071219",
              thisDate    = idNumber.substring(0,6),
              pbaDate     = pba.idNumber.substring(0,6);
        boolean thisLastCentury = thisDate.compareTo(today) > 0,
              pbaLastCentury  = pbaDate.compareTo(today) > 0;
        if ( thisLastCentury == pbaLastCentury )
            return pbaDate.compareTo( thisDate );
        else if (thisLastCentury)
            return 1;
        else
            return -1;
    } // compareTo

    public static String toString( Set<Person> sp ) {
        SortedSet<PersonByAge> ssp = new TreeSet<PersonByAge>();
        String res = "";
        for ( Person p : sp )
            ssp.add( new PersonByAge( p.getId(), p.getName() ) );
        for (PersonByAge pba : ssp )
            res = res + pba.getId() + " " + pba.getName() + "\n";
        return res;
    } // toString
}
```

Och ett litet testprogram som inte ingick i uppgiften:

```
public static void main(String[] args) {
    HashSet<Person> hsp = new HashSet<Person>();
    hsp.add(new Person( "460702-xxxx", "Kalle" ));
    hsp.add(new Person( "560702-xxxx", "Kajsa" ));
    hsp.add(new Person( "040702-xxxx", "Knatte" ));
    hsp.add(new Person( "050702-xxxx", "Fnatte" ));
    hsp.add(new Person( "050703-xxxx", "Tjatte" ));

    System.out.println( toString(hsp));

} // main

} // class PersonByAge
```

Vilket ger vid exekvering:

```
> javac PersonByAge.java
> java PersonByAge
050703-xxxx Tjatte
050702-xxxx Fnatte
040702-xxxx Knatte
560702-xxxx Kajsa
460702-xxxx Kalle
```

```
Uppg 5: a) public class IncAndDecrOneMillion {

    public static void main( String[] args ) {

        MyThread mt1 = new MyThread( 1, 1000000 );
        MyThread mt2 = new MyThread( -1, 1000000 );
        mt1.start();
        mt2.start();
        try {
            mt1.join();
            mt2.join();
        }
        catch( InterruptedException ie) { }

        System.out.println( MyThread.shared );

    } // main
} // class IncAndDecrOneMillion
```

b) Det beror på att koden kan bli 'saxad' (interleaved)

c) De enda ändringarna som behövs är merkerade med `/**/`

```
public class MyThread extends Thread {

    private int whatToAdd;
    private int howMany;

    public static int shared = 0;
    public static Object lockObj
        = new Object();          /**/

    public MyThread( int toAdd, int times ) {
        whatToAdd = toAdd;
        howMany   = times;
    } // constructor MyThread

    public void run() {
        int temp = 0;
        for ( int i = 0; i < howMany; i++ ) {
            synchronized(lockObj) {          /**/
                temp  = shared + whatToAdd;
                shared = temp;
            }                                /**/
        }
    } // run

} // class MyThread
```

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

        int iterMt1 = Integer.parseInt( args[0] );
        int iterMt2 = Integer.parseInt( args[1] );

        MyThread mt1 = new MyThread( 1, iterMt1 );
        MyThread mt2 = new MyThread( -1, iterMt2 );
    }
```

Resten som tidigare.