Institutionen för
Datavetenskap
CTH, GU

HT07
TDA550, DIT720
07-12-18

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 subklass 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, subklass 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 NoSuchElemetException om kön är tom.
     */
    public E dequeu();

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

    /**
     * Returnera antalet element i kön.
     */
    public int size();

} // inteface 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 subklass 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 subklass 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 konstruktor **(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 deklareras 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)**