Tentamen för TDA550 **Objektorienterad programvaruutveckling IT, fk**

DAG: 10-04-06 TID: 8:30 – 12:30

Ansvarig: Christer Carlsson, ankn 1038

Förfrågningar: Christer Carlsson

Resultat: erhålls via Ladok

Betygsgränser:

3:a	24 poäng
4:a	36 poäng
5:a	48 poäng
maxpoäng	60 poäng

Siffror inom parentes: anger maximal poäng på uppgiften.

Granskning: Tisdag 27/4 kl 11-13 och onsdag 28/4 kl 12-13, rum 6128 i EDIT-huset.

Hjälpmedel: Skansholm: Java direkt + utdelat extrakapitel (eller motsvarande lärobok Java)

Var vänlig och: Skriv tydligt och disponera papperert på lämpligt sätt.

Börja varje uppgift på nytt blad. Skriv ej på baksidan av papperet.

Observera: Uppgifterna är ej ordnade efter svårighetsgrad. Titta därför igenom hela tentamen innan du börjar skriva.

Alla program skall vara välstruktruerade, lätta att överskåda samt enkla att förstå.

Vid rättning av uppgifter där programkod ingår bedöms principella fel allvarligare än smärre språkfel.

LYCKA TILL!!!!

Uppgift 1.

a) Vad innebär det att en klass är *icke-muterbar* (*immutable*)? Redogör för vilka fördelar det finns med icke-muterbara klasser.

(3 poäng)

b) Givet följande klasser:

```
public class A {
    public A() {
        System.out.println("A.A()");
    }
    public void f(A a) {
        System.out.println("A.f(A)");
    }
    public void f(B b) {
        System.out.println("A.f(B)");
    }
    public void g(A a) {
    System.out.println("A.g(A)");
    }
}
public class B extends A {
    public B() {
        System.out.println("B.B()");
    }
    public void f(A a) {
        System.out.println("B.f(A)");
    }
    public void f(B a) {
        System.out.println("B.f(B)");
    }
    public void g(B b) {
        System.out.println("B.g(B)");
    }
    public static void h(A a) {
        System.out.println("B.h(A)");
    }
}
```

Vad blir resultatet *för var och en av* följande satser (ger kompileringsfel, ger exekveringsfel, skriver ut xxx, etc)?

```
A a =  new A();
B b = new B();
A ab =  new B();
a.f(ab);
a.g(b);
b.f(ab);
b.g(a);
b.h(ab);
ab.f(a);
ab.h(a);
ab.f(ab);
```

(6 poäng)

Uppgift 2.

a) Givet följande kod

```
public static void method(int y) {
    try {
        System.out.print("A");
        int x = 1 / y ;
        System.out.print("B");
    }
    catch (ArithmeticException e) {
        System.out.print("C");
    }
    finally {
        System.out.print("D");
    }
}
```

Vad kommer att skrivas ut för följande metodanrop?

i) method(1)
ii) method(0)

(3 poäng)

b) Betrakta nedanstående interface:

```
public interface Modem {
    public void dial(String pno);
    public void hangUp();
    public boolean isConnected();
    public void send(char[] c);
    public void receive(char[] c);
}
```

Varför bör detta interface delas upp i två interface? Hur skall dessa båda interface se ut?

(4 poäng)

c) Skriv om nedanstående klass så att den blir generisk och därmed kan användas för att handha andra klasser än bara Integer.

```
import java.util.*;
public class MyDS {
    private List<Integer> data;
    public MyDS() {
        data = new ArrayList<Integer>();
    }

    public Integer first() {
        return data.get(0);
    }

    public void append(Integer val) {
        data.add(val);
    }
    // fler metoder som vi inte bryr oss om i denna uppgift
}
```

(3 poäng)

Uppgift 3.

Koden i metoden doSwitch är mindre bra.

```
public static void doSwitch(Object obj) {
    if (o instanceof A) {
        ((A) obj).doIt();
    } else if ( obj instanceof B) {
        ((B) obj).doIt();
    } else if ( obj instanceof C) {
        ((C) obj).doIt();
    }
}
public class A {
    public void doIt() {
        System.out.println("This is A");
    }
}
public class B {
    public void doIt() {
        System.out.println("This is B");
    }
}
public class C {
    public void doIt() {
        System.out.println("This is C");
    }
}
```

a) Motivera vad som är problemet.

b) Lös problemet. Om du har flera lösningsalternativ, motivera varför du valt den lösning du gjort. Du får ändra fritt i koden bara resultatet blir som det ursprungliga (d.v.s. klasserna skall finnas och anropet doSwitch(...) skall producera samma utskrift).

(8 poäng)

Uppgift 4.

Kalle Klåpare har skrivit ett Javaprogram som använder två trådar, thread1 och thread2. Nu vill han att thread1 ska kunna skicka över en sträng till thread2. Han har därför skrivit följande klass för att hantera överföringen:

```
public class Buffer {
    private String message;
    public void set(String m) {
        message = m;
    }
    public String get() {
        String m = message;
        message = null;
        return m;
    }
}
```

thread1 anropar då och då set() för att lägga in en sträng i bufferten. thread2 anropar hela tiden get() och när något annat än **null** returneras utför thread2 någon typ av operation på den mottagna strängen. Tyvärr fungerar Kalles program väldigt dåligt.

a) Ge ett exempel på problem Kalle kan råka ut för med lösningen ovan.

b) Skriv om klassen Buffer ovan så att den blir både trådsäker *och* effektiv.

(8 poäng)

Uppgift 5.

I ett program som hanterar aritmetiska uttryck finns nedanstående två klasser som är identiska så när som på tre rader.

```
public class Add implements Expr {
    private Expr expr1, expr2;
    public Add(Expr expr1, Expr expr2) {
        this.expr1 = expr1;
        this.expr2 = expr2;
    }
    public String toString() {
        StringBuffer buffer = new StringBuffer();
        buffer.append("(").append(expr1);
        buffer.append('+');
        buffer.append(expr2).append(")");
        return buffer.toString();
    }
    // omissions
}

public class Mul implements Expr {
    private Expr expr1, expr2;
    public Mul(Expr expr1, Expr expr2) {
        this.expr1 = expr1;
        this.expr2 = expr2;
    }
    public String toString() {
        StringBuffer buffer = new StringBuffer();
        buffer.append("(").append(expr1);
        buffer.append("*");
        buffer.append(expr2).append(")");
        return buffer.toString();
    }
    // omissions
}
```

Använd *Template*-eller *Strategy*-mönstret för att eliminera duplicerad kod och visa koden för de klasser som ersätter de befintliga.

(7 poäng)

Uppgift 6.

Du ska skriva klassen Synonyms som skall användas för att lagra synonymer, d.v.s. ord med likartad betydelse. Låt varje ord vara en nyckel i en HashMap och lagra ordets synonymer i en ArrayList som värde till nyckeln. Själva synonymordlistan skall allså i implementationen lagras i ett objekt av typen HashMap<String, ArrayList<String>>.

Följande metoder ska finnas i klassen:

public void add(String word1, String word2)	Lägger in word1 som synonym till word2 och word2 som synonym till word1.
public void remove(String word1, String word2)	Tar bort word1 som synonym till word2 och word2 som synonym till word1.
public String toString()	Returnerar en sträng på formen ”Det finns X ord med tillhörande synonymer lagrade” där X ersätts av antalet lagrade ord.
public List<String> getSynonyms(String word)	Returnerar synonymerna till ordet word som en List.

Du behöver inte ta hänsyn till stora och små bokstäver i orden, utan kan anta att endast små bokstäver används.

(Utdrag från APIn för klasserna HashMap finns som bilaga sist i denna tes.)

(10 poäng)

Uppgift 7.

I klassen Collections finns den statiska metoden

<T> **void** sort(List<T> list, Comparator<? **super** T> c)
Sorts the specified list according to the order induced by the specified comparator.

Du skall nu använda denna metod för att skriva en metod

public static void printSynonyms(String word, List<String> synomyms)

som får ett ord word och en lista synonyms med ordets synonymer. Metoden skall skriva ut ordet på System.out tillsammans med ordets synonymer enligt:

Synonymer till kunna: veta, förstå, känna till, ha kännedom om

Synonymerna skall alltså skrivs ut sorterade; i första hand efter deras längd och i andra hand (d.v.s. om två synonymer är lika långa) efter alfabetisk ordning (d.v.s. den ordning som metoden compareTo ger för klassen String). Om ett ord inte har några synonymer (d.v.s. listan synonymer är **null** eller har längden 0) skall utskriften bli enligt:

Ordet pannkaka finns ej.

För att kunna implementera metoden printSynonyms, måste du således skapa en klass som implementerar interfacet Comparator och realiserar metoden compare enligt beskrivningen ovan. Interfacet Comparator har följande utseende:

```
public interface Comparator<T> {
    // Compares its two arguments for order.
    int compare(T o1, T o2);
    // Indicates whether some other object is "equal to" this comparator.
    boolean equals(Object obj);
}
```

Du skall *inte* överskugga metoden equals.

(8 poäng)

LÖSNINGSFÖRSLAG - 100406

Uppgift 1.

a) Objekten som tillhör en icke-muterbar klass är oförändliga, dvs de behåller under hela sin livstid det tillstånd som de fick när de skapades. Icke-muterbara objekt har många fördelar:

- De är lätta att förstå.
- De är alltid trådsäkra.
- Det är ofarligt att lämna ut referenser till dem. Det gör inget om en massa olika objekt råkar använda ett och samma immutable object.
- Deras interna data kan återanvändas. Om ett nytt immutable object ska skapas där vissa fält inte skiljer sig från de i ett befintligt objekt av samma klass kan alla oförändrade fält referera till exakt samma instanser som i det befintliga objektet.
- Det är lätt att använda immutable objects som tillstånd i andra objekt.
- Det är ofarligt att skicka dem till andra processer i distribuerade system.

b)

A a = **new** A();	Skriver ut:	A.A()
B b = **new** B();	Skriver ut:	A.A()
		B.B()
A ab = **new** B();	Skriver ut:	A.A()
		B.B()
a.f(ab);	Skriver ut:	A.f(A)
a.g(b);	Skriver ut:	A.g(A)
b.f(ab);	Skriver ut:	B.f(A)
b.g(a);	Skriver ut:	A.g(A)
b.h(ab);	Skriver ut:	B.h(A)
ab.f(a);	Skriver ut:	B.f(A)
ab.h(a);	Ger kompileringsfel, metoden h finns inte klassen A	
ab.f(ab);	Skriver ut:	B.f(A)

Uppgift 2.

a)

i) ABD

ii) ACD

b)

Interfacet har två ansvarsområden och strider mot S*ingle Responsibility Principle* .

```
public interface Contact {
    public void dial(String pno);
    public void hangup();
    public boolean isConnected();
}
public interface Transmit {
    public void send(char[] c),
    public void recive(char[] c);
 }
```

c)

```
import java.util.*;
public class MyDS<T> {
    private List<T> data;
    public MyDS() {
        data = new ArrayList<T>();
    }

    public T first() {
        return  data.get(0);
    }

    public void append(T val) {
        data.add(val);
    }
    // fler metoder som vi inte bryr oss om i denna uppgift
}
```

Uppgift 3.

a) Om man skulle införa ytterligare en klass måste man ändra i metoden doSwitch.

b) Inför ett gränssnitt som klasserna A, B och C implementerar. Alternativt kan man införa en abstrakt klass som klasserna A, B och C utökar. Gränssnitt är att föredra eftersom klasser endast kan ärva från en klass, men kan implementera ett godtyckligt antal interface.

// Inför ett interfac

```
public interface I {
    public void doIt();
}
public class A implements I {
    public void doIt() {
        System.out.println("This is A");
    }
}
public class B implements I {
    public void doIt() {
        System.out.println("This is B");
    }
}
public class C implements I {
    public void doIt() {
        System.out.println("This is C");
    }
}
public static void doSwitch(I obj) {
    obj.doIt();
}
```

// Inför en abstrakt klass

```
public abstract class Abst {
     public abstract void doIt();
}

public class A extends Abst {
    public void doIt() {
        System.out.println("This is A");
    }
}
public class B extends Abst  {
    public void doIt() {
        System.out.println("This is B");
    }
}
public class C extends Abst  {
    public void doIt() {
        System.out.println("This is C");
    }
}
public static void doSwitch(Abst obj) {
    obj.doIt();
}
```

Uppgift 4.

a) Ett problem Kalle kan få är att meddelanden försvinner eftersom set() skulle kunna anropas två gånger i följd av samma tråd. Ett annat problem som kan uppstå är att den läsande tråden stjäl all tillgänglig CPU-tid (busy-wait) vilket kan göra att andra trådar, t.ex. den som anropar set() aldrig får köra.

b)

```
public class Buffer {
   private String message;
   public synchronized void set(String m) {
      while (m != null) {
         try {
            wait();
         } catch(InterruptedException e) {}
      }
      message = m;
      notifyAll();
   }
   public synchronized String get() {
      while (message == null) {
         try {
            wait();
         } catch(InterruptedException e) {}
      }
      String m = message;
      message = null;
      notifyAll();
      return m;
   }
}
```

Uppgift 5.

Template-mönstret:

```
public abstract class ArithmeticExpr implements Expr {
    private Expr expr1, expr2;
    protected ArithmeticExpr(Expr expr1, Expr expr2) {
        this.expr1 = expr1;
        this.expr2 = expr2;
    }
    protected abstract String getOpString();
    public String toString() {
        StringBuffer buffer = new StringBuffer();
        buffer.append("(").append(expr1);
        buffer.append(getOpString());
        buffer.append(expr2).append(")");
        return buffer.toString();
    }
    // omissions
}
public class Add extends ArithmeticExpr {
    public Add(Expr expr1, Expr expr2) {
        super(expr1, expr2);
    }
    protected String getOpString() {
        return "+";
    }
    // omissions
}
public class Mul extends ArithmeticExpr {
    public Mul(Expr expr1, Expr expr2) {
        super(expr1, expr2);
    }
    protected String getOpString() {
        return "*";
    }
    // omissions
}
```

Strategimönstret:

```
public interface OperationStrategy {
    public String oper();
}
public class Add implements OperationStrategy {
    public String oper() {
        return "+";
    }
}
public class Mul implements OperationStrategy {
    public String oper() {
        return "-";
    }
}
public class ArithmeticExpr implements Expr {
    private Expr expr1, expr2;
    private OperationStrategy strategy;
    protected ArithmeticExpr(OperationStrategy strategy, Expr expr1, Expr expr2) {
        this.strategy = strategy;
        this.expr1 = expr1;
        this.expr2 = expr2;
    }
    public String toString() {
        StringBuffer buffer = new StringBuffer();
        buffer.append("(").append(expr1);
        buffer.append(strategy.oper());
        buffer.append(expr2).append(")");
        return buffer.toString();
    }
    // omissions
}
```

Uppgift 6.

```
import java.util.*;
public class Synonyms {
    private Map<String, ArrayList<String>> ordbank = new HashMap<String, ArrayList<String>>();
    public void add(String word1, String word2) {
        addSynonyms(word1, word2);
        addSynonyms(word2, word1);
    }//add

    private void addSynonyms(String word, String synonym) {
        ArrayList<String> synonymer = ordbank.get(word);
        if (synonymer == null) {
            synonymer = new ArrayList<String>();
            synonymer.add(synonym);
            ordbank.put(word,synonymer);
        }
        else if (!synonymer.contains(synonym)){
            synonymer.add(synonym);
        }
    }//addSynonymers

    public void remove(String word1, String word2) {
        removeSynonyms(word1, word2);
        removeSynonyms(word2, word1);
    }//remove

    private void removeSynonyms(String word, String synonym) {
        ArrayList<String> synonymer = ordbank.get(word);
        if (synonymer != null)
            synonymer.remove(synonym);
    }//removeSynomymers

    public List<String> getSynonyms(String word) {
        return ordbank.get(word);
    }//getSynonyms

    public String toString() {
        return ordbank.size()+" ord med synonymer";
    }//toString
}//Synonyms
```

Uppgift 7.

```
import java.util.*;
public class SortSynonyms implements Comparator<String> {
    public int compare(String s1, String s2) {
        int len1 = s1.length();
        int len2 = s2.length();
        if  (len1 != len2)
            return len1 - len2;
        else
        return s1.compareTo(s2);
    } //compare
}//SortSynonyms

public static void printSynonyms(String word, List<String> synonyms) {
    if (synonyms == null || synonyms.size() == 0)
        System.out.println("Ordet " + word + " finns ej");
    else {
        SortSynonyms sort = new SortSynonyms();
        Collections.sort(synonyms, sort);
        String res = "Synonymer till " + word + ": ";
        for (int i = 0; i < synonyms.size(); i++) {
            res = res + synonyms.get(i);
            if (i < synonyms.size()-1)
                res = res + ", ";
        }
        System.out.println(res);
    }
}//printSynonyms
```