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

DAG: 15-08-19 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: Måndag 21/9 kl 12-13 och tisdag 22/9 kl 12-13, rum 6128 i EDIT-huset.

Hjälpmedel: Inga hjälpmedel är tillåtna förutom bilagan till tesen.

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.

Betrakta nedanstående klasser och interface:

```
public interface A {
   public void a(double x);
}//A

public interface B {
   public void b();
}//B

public abstract class C implements A {
   public C() {
      System.out.println("constructor i C");
   }//constructor

   public void a(double x) {
      System.out.println("a() in C");
   }//a
   public abstract void c();
}//C

public class D extends C  {
   public D() {
      System.out.println("constructor i D");
   }//constructor

   public void c() {
      System.out.println("c() in D");
   }//c
}//D
```

```
public class E extends D implements B {
   public E() {
      System.out.println("constructor i E");
   }//constructor

   public void a(double i) {
      System.out.println("a() in E");
   }//a

   public void b() {
      System.out.println("b() in E");
   }//b
}//E

public class F extends D {
   public F() {
      System.out.println("constructor i F");
   }//constructor

   public void a(int x) {
      System.out.println("a() in F");
   }//a
}//F
```

a) Rita ett UML-diagram över klasserna och gränssnitten. (1 poäng)

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

i)
```
A x = new C();
x.a(5);
```

ii)
```
A x = new E();
x.a(5);
```

iii)
```
A x = new F();
x.a(5);
```

iv)
```
B x = new E( );
D y = (D) x;
y.c();
```

v)
```
A x = new D( );
E y = (E) x;
y.b();
```

vi)
```
D x = new E( );
B y = (B) x;
y.b();
```

(6 poäng)

Uppgift 2.

Ibland är det viktigt att en klass endast instansieras *en gång* under en programexekvering. Till exempel kan det i ett simuleringsprogram vara direkt olämpligt att flera slumptalsgeneratorer existerar samtidigt. Tillämpa designmönstret Singleton så att man kan hantera slumptal på följande sätt

```
SingletonRandom r1 = SingletonRandom.getInstance();
SingletonRandom r2 = SingletonRandom.getInstance();
int s1 = r1.nextInt(10);
int s2 = r2.nextInt(20);
...
```

Variablerna r1 och r2 skall förstås referera till ett och samma objekt, så alla anrop av nextInt drar pseudoslumptal ur samma sekvens. Implementera SingletonRandom. Utnyttja klassen java.util.Random.

(5 poäng)

Uppgift 3.

Betrakta nedanstående klasser:

```
public class MusicPlayer {
    private Object o;
    private Volume vol = new Volume();
    public MusicPlayer(Object o) {
        this.o = o;
    }
    public void lower() {
        vol.lower();
    }
    public void higher() {
        vol.higher();
    }
    public void play() {
        if (o instanceof Trumpet){
            ((Trumpet) o).playTrumpet();
        } else if (o instanceof Piano){
            ((Piano) o).playPiano();
        } else if (o instanceof Guitar){
            ((Guitar) o).playGuitar();
        }
    }
    public void changeInstrument(Object ot) {
        this.o = o;
    }
}//MusicPlayer

public class Trumpet {
    public void playTrumpet() {
        //code for playing trumpet
    }
    //constructors and methoder not shown here
}//Trumpet

public class Piano {
    public void playPiano() {
        //code for playing piano
    }
    //constructors and methoder not shown here
}//Piano

public class Guitar {
    public void playGuitar() {
        //code for playing guitar
    }
    //constructors and methoder not shown here
}//Guitar
```

Designen strider mot *Open-Closed Principle*. Åtgärds detta genom att nyttja designmönstret Strategy.

(6 poäng)

Uppgift 4.

a) Förklara vad det innebär att en typ Sub är en äkta subtyp till typen Sup. (2 poäng)

b) Betrakta specifikationen för de två interfacen nedan:

```
// A list of ints where each int is at a position where the first
// position is 0, second position is 1, etc.
interface IntList {
    @requires       receiver has >= i elements
    @effects        puts x at position i, moving all elements at positions >= i one position higher
    @modifies       this
     void insertAt(int x, int i);

    @effects        puts x at some unspecified position i, moving all elements at positions >= i one position higher
    @modifies       this
     void insert(int x);

    @requires       receiver has > i elements
    @returns        the int currently at position i
     int get(int i);
}


// A sorted list of ints where each int is at a position where the first
// position is 0, second position is 1, etc. Ints are in increasing order.
interface SortedIntList {
    @requires       receiver has >= i elements and the effect of the operationcmaintains a sorted list (i.e., i is a
                    "legal" position to add x)
    @effects        puts x at position i, moving all elements at positions >= i one position higher.
    @modifies       this
     void insertAt(int x, int i);

    @effects        puts x at some position i that maintains sorted order (moving all elements previously at position
                    i or greater to one position higher).
    @modifies       this
     void insert(int x);

    @requires       receiver has > i elements.
    @returns        the int currently at position i.
     int get(int i);
}
```

i) Är SortedIntList en äkta subtyp till IntList? Motivera ditt svar! (2 poäng)

ii) Är IntList en äkta subtyp til SortedIntList? Motivera ditt svar! (2 poäng)

Uppgift 5.

Vid användning av designmönstret Observer kan man ha både flera observerade objekt och flera observatörer. En observatör kan t.ex. observera två olika objekt. Antag att vi har klasserna

```
public class FirstClass extends Observable { ... }
public class SecondClass extends Observable { ... }
```

Antag att både FirstClass och SecondClass kan anropa notifyObservers med en sträng eller ett heltal som argument.

Nedan ges ett kodskelett för klassen Monitor:

```
import java.util.Observer;
import java.util.Observable;
public class Monitor implements Observer {
    public Monitor(FirstClass a, SecondClass b) {
        a.addObserver(this);
        b.addObserver(this);
    }
    public void update(Observable o, Object arg) {
        //uttgift att färdigställa
    }
}
```

Din uppgift är att implementera metoden update i klassen Monitor. Metoden update skall fungera enligt följande:

Om anropet orsakades av att FirstClass anropade notifyObservers med en sträng som argument så skall strängen översättas till stora bokstäver och skrivas ut.

Om istället anropet orsakades av att SecondClass anropade notifyObservers med ett heltal som argument så skall talet multiplicerat med 10 skrivas ut.

Annars görs ingenting.

Anm: FirstClass- och SecondClass-objekten kan inte nås via instansvariabler. (4 poäng)

Uppgift 6.

Följande interface och klass är givna:

```
public interface Sorter {
    public void sort(int[] numbers);
}
import java.awt.List;
public class NumberSorter {
    public void sort(List<Integer> numbers) {
        Collections.sort(numbers);
    }
}
```

Skriv en klass SortListAdapter som implementerar desigmönstret *Adapter* för att anpassa gränssnittet för typen NumberSorter till typen Sorter. (5 poäng)

Uppgift 7.

Betrakta nedanstående kod:

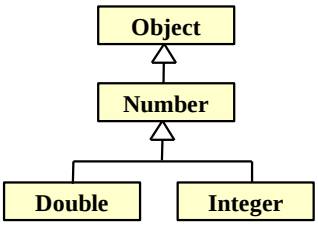
```
public interface SomeInterface {
     void someMethod();
}//SomeInterface

public class SomeClass implements SomeInterface {
    void someMethod() {
        //some code
    }
}//SomeClass
```

Förklara varför klassen SomeClass inte går att kompilera. (2 poäng)

Uppgift 8.

Betrakta nedanstående deklarationer:

```
List<Object> listObject;
List<Number> listNumber;
List<? extends Number> listExtendsNumber;
List<? super Number> listSuperNumber;
List<Double> listDouble;
List<? extends Double> listExtendsDouble;
List<? super Double> listSuperDouble;
Number n;
Double d;
Integer i;
```



Ange för var och en av följande satser om satsen är korrekt eller ger kompileringsfel:

a) listNumber.add(d);
b) listExtendsNumber.add(n);
c) listSuperNumber.add(i);
d) i = listExtendsNumber.get(0);
e) n = listExtendsNumber.get(0);
f) n = listSuperNumber.get(0);
g) listNumber = listDouble;
h) listExtendsNumber = listDouble;
i) listExtendsNumber = listExtendsDouble;
j) listSuperDouble = listObject;

(5 poäng)

Uppgift 9.

Betrakta nedanstående interface och klasser:

```
public interface Airplane {
    public void construct();
}//Airplane

public class OriginalAirplane implements Airplane {
    @Override
    public void construct() {
        System.out.print("Original Airplane. ");
    }
}//OriginalAirplane
```

Använd designmönstret *Decorator* för att skapa två nya klasser EjectionSeatDecorator och TurboDecorator som dekorerar objekt av klassen Airplane med en katapultstol respektive en turbomotor.

Nedanstående main-metod illustrerar hur det hela är tänkt att fungera:

```
public static void main(String[] args) {
    Airplane planeA = new OriginalAirplane();
    planeA.construct();
    System.out.println();
    Airplane planeB = new EjectionSeatDecorator(new OriginalAirplane());
    planeB.construct();
    System.out.println();
    Airplane planeC = new EjectionSeatDecorator(new TurboDecorator (new OriginalAirplane()));
    planeC.construct();
    System.out.println();
} //main
```

Utskriften från main-metoden skall bli:

```
Original Airplane.
Original Airplane. Inserting Ejection Seat to airplane.
Original Airplane. Inserting Turbo to airplane. Inserting Ejection Seat to airplane.
```

Tips: För att undvika duplicering av kod skall du skapa en abstrakt klass AirplaneDecorator som klasserna EjectionSeatDecorator och TurboDecorator ärver från. (6 poäng)

Uppgift 10.

a) Betrakta nedanstående klasser:

```
public class Print implements Runnable {
   private String str;
   public Print(String s) {
      str = s;
   }
   public void run(){
      System.out.println(str + ". ");
   }
}//Print
```

```
import java.util.*;
public class PrintTest {
   public static void main(String[] args) {
      Thread t1 = new Thread(new Print("Thread1"));
      Thread t2 = new Thread(new Print("Thread2"));
      Thread t3 = new Thread(new Print("Thread3"));
      t3.start();
      t2.start();
      t1.start();
   }
}//PrintTest
```

Vilka möjliga utskrifter kan erhållas då main-metoden i klassen PrintTest exekveras? (3 poäng)

b. Betrakta klassen WaterTank, nedan, som används för att handha en vattentank:

```
public class WaterTank {
   private double level;
   public void fill(double volume) {
      level = level + volume;
   }
   public boolean empty(double volume) {
      if (volume < level) {
         double newLevel = level - volume;
         level = newLevel;
         return true;
      }
      else
         return false;
   }
}//WaterTank
```

i) Klassen är inte trådsäker. Förklara med ett *exempel* varför! (2 poäng)

ii) Skriv om klassen så att den blir trådsäker! (2 poäng)

Uppgift 11.

Nedan finns ett skelett till klassen AppStore, som skall används för att hålla reda på vilka appar som finns tillgängliga för olika operativsystem:

```
public class AppStore {
    private Map<String, Set<String>> store;
    public AppStore() {
        //skall implementeras i deluppgift a)
    }
    public void addApp(String osType, String appName) {
        //skall implementeras i deluppgift b)
    }
    public String getOsTypeWithMaxNumApps() {
        //skall implementeras i deluppgift c)
    }
}//AppStore
```

Din uppgift är att färdigställa klassen.

a) Implementera konstruktorn, som skapar en tom Map. (1 poäng)

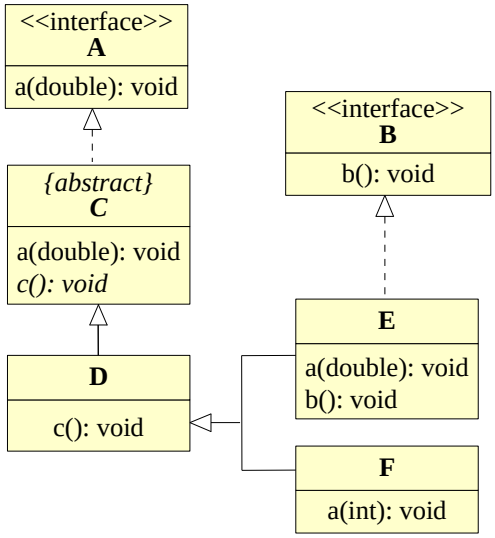
b) Implementera metoden addApp, vilken lägger till appen appName till operativsystemet osType. (3 poäng)

c) Implementera metoden getOsTypeWithMaxNumApps, vilken returnerar det operativsystem som har flest appar tillgängliga. Du får anta att inga operativsystem har samma antal applikationer. (3 poäng)

Tentamen 150819 - LÖSNINGSFÖRSLAG

Uppgift 1.

a)



b)

i) Tilldelningen A x = **new** C() ger kompileringsfel eftersom klassen C är abstrakt.

ii) Ger utskriften:

constructor i C
constructor i D
constructor i E
a() in E

iii) Ger utskriften:

constructor i C
constructor i D
constructor i F
a() in C

iv) Ger utskriften:

constructor i C
constructor i D
constructor i E
c() in D

v) Ger utskriften:

constructor i C
constructor i D
sedan inträffar ett exekveringsfel pga att den dynamiska typen på x är D och typen D kan inte typomvandlas till typen E.

iv) Ger utskriften

constructor i C
constructor i D
constructor i E
c() in E

Uppgift 2.

```
public class SingletonRandom {
    private static SingletonRandom instance = null;
    private Random random;

    private SingletonRandom() {
        random = new Random();
    }//constructor

    public static synchronized SingletonRandom getInstance() {
        if (instance == null)
            instance = new SingletonRandom();
        return instance;
    }//getInstance

    public int nextInt(int limit) {
        return random.nextInt(limit);
    }//nextInt

}//SingeltonRandom
```

Uppgift 3.

```
public class MusicPlayer {
    private Instrument instrument;
    private Volym vol = new Volym();
    public Musicplayer(Instrument instrument) {
        this.instrument = instrument;
    }
    public void lower() {
        vol.lower();
    }
    public void higher() {
        vol.higher();
    }
    public void play() {
        instrument.play();
    }
    public void changeInstrument(Instrument theInstrument) {
        instrument = theInstrument;
    }
}// MusicPlayer

public interface Instrument {
    public abstract void play();
}//Instrument

public class Trumpet implements Instrument {
    public void play() {
        //code for playing trumpet
    }
    //constructors and methoder not shown here
}//Trumpet

public class Piano implemens Instrument {
    public void play() {
        //code for playing piano
    }
    //constructors and methoder not shown here
}//Piano

public class Guitar implements Instrument {
    public void play() {
        //code for playing guitar
    }
    //constructors and methoder not shown here
}//Guitar
```

Uppgift 4.

a) Begreppet *äkta subtyp* definieras av *Liskov Substitution Principle*. Att Sub är en äkta subtyp till Sup innebär att man i ett program kan byta ut ett objekt av klassen Sup mot ett objekt av typen Sub utan att beteendet hos programmet förändras. Detta innebär att en äkta subtyp inte kan ha en svagare specifikation. Förlitar sig en klient på supertypens specifikation och subtypen har en svagare specifikation kommer det att gå galet.

b)

i) Nej! Specifikationen för metoden insertAt i klassen SortedIntList är svagare än i klassen IntList. Detta pga att förvillkoret är starkare. Förvillkoret kräver att klienten väljer ett värde på parametern i som bevarar den sorterade ordningen i listan.

ii) Nej! Specifikationen för metoden insert i klassen IntList är svagare än i klassen SortedIntList. Detta pga att eftervillkoret är svagare, då det inte finns några garantier på var det insatta element hamnar i listan.

Uppgift 5.

```
public void update(Observable o,Object arg) {
    if (o instanceof FirstClass && arg instanceof String) {
        String s = (String) arg;
        System.out.println(s.toUpperCase());
    }
    if (o instanceof SecondClass && arg instanceof Integer) {
        Integer i = (Integer) arg;
        System.out.println(i*10);
    }
}//update
```

Uppgift 6.

```
import java.awt.List;
public class SortListAdapter implements Sorter {
    private NumberSorter sorter = new NumberSorter();

    @Override
    public void sort(int[] numbers) {
        List<Integer> numberList = new ArrayList<Integer>();
        for (int i = 0; i < numbers.length; i++)
            numberList.add(numbers[i]);
        sorter.sort(numberList);
        for (int i = 0; i < numbers.length; i++)
            numbers[i] = numberList.get(i);
    }
}//SortListAdapter
```

Uppgift 7.

Metoderna i ett interface är alltid publika och abstrakta, oberoende av om detta anges eller inte. Deklarationen av vårt interface ser alltså egentligen ut enligt:

```
public interface SomeInterface {
    public abstract void someMethod();
}
```

En klass som implementerar interfacet kan inte begränsa synligheten av metoderna som definieras av interfacet, vilket klassen SomeClass gör. En korrekt implementation av klassen är:

```
public class SomeClass implements SomeInterface {
    public void someMethod() {
        //some code
    }
}
```

Uppgift 8.

a) Korrekt. I en list av typen List<Number> kan ett element av typen Double läggas in eftersom Double är en suptyp till Number.

b) Kompileringsfel. List<? **extends** Number> kan antingen vara List<Number>, List<Double> eller List<Integer>. Kompilatorn vet inte vilken den specifika är och i typen List<Double> eller List<Integer> kan man inte lägga in ett objekt av typen Number.

c) Korrekt. List<? **super** Number> kan antingen vara List<Number> eller List<Object> och Integer är subtyp både till Number och Object.

d) Kompileringsfel. Typen List<? **extends** Number> inkluderar typerna List<Number> samt List<Double>, och varken Number eller Double är subtyper med typen Integer.

e) Korrekt. I alla listor av typen List<? **extends** Number> (List<Number>, List<Double> eller List<Integer>) är objekten kompatibla med typen Number.

f) Kompileringsfel. List<? **super** Number> kan antingen vara List<Number> eller List<Object>. Kompilatorn vet inte vilken den specifika är och i typen List<Object> är inte elementen kompatibla med typen Number.

g) Kompileringsfel. List<Double> är inte en subtyp till List<Number>, utan en subtyp till Collection<Double>.

h) Korrekt. List<? **extends** Number> kan antingen vara List<Number>, List<Double> eller List<Integer>, samt List<SomeType> där SomeType är en subtyp till någon av typerna Double eller Integer..

i) Korrekt. Se deluppgift h).

j) Korrek. List<? **extends** Double> kan antingen vara List<Double>, List<Number> eller List<Object>.

Uppgift 9.

```
public abstract class AirplaneDecorator implements Airplane {
    protected Airplane decoratedAirplane;
    public AirplaneDecorator(Airplane decoratedAirplane) {
        this.decoratedAirplane = decoratedAirplane;
    }
    public void construct() {
        decoratedAirplane.construct();
    }
}//AirplaneDecorator

public class EjectionSeatDecorator extends AirplaneDecorator {
    public EjectionSeatDecorator(Airplane decoratedAirplane) {
        super(decoratedAirplane);
    }
    @Override
    public void construct() {
        decoratedAirplane.construct();
        System.out.print(" Inserting Ejection Seat to airplane. ");
    }
}//EjectionSeatDecorator

/public class TurboDecorator extends AirplaneDecorator {
    public TurboDecorator(Airplane decoratedAirplane) {
        super(decoratedAirplane);
    }
    @Override
    public void construct() {
        decoratedAirplane.construct();
        System.out.print(" Inserting Turbo to airplane. ");
    }
}//TurboDecorator
```

Uppgift 10.

a) 6 olika utskrifer är möjliga, nämligen alla permutationer av strängarna "Thread1", "Thread2" och "Thread3", dvs

Thread1. Thread2. Thread3.
Thread1. Thread3. Thread2.
Thread2. Thread1. Thread3.
Thread2. Thread3. Thread1.
Thread3. Thread1. Thread2.
Thread3. Thread2. Thread1.

b)

i) Det kan uppstå ett s.k. *race condition*, eftersom t.ex. en tråd T1 kan anropa metoden empty() samtidigt som en annan tråd T2 anropar metoden fill(). Trådarna konkurrerar då om variabeln level. Anta att level har värdet 100.0. Följande kan då inträffa:

T1 vill tömma ur volymen 50. Eftersom level har värdet 100 går detta bra och T1 påbörjar beräkningen av satsen

```
double newLevel = level - volume;
```

Under vill T2 fylla tanken med volymen 100. T2 hinner utföra satsen

```
level = level + volume;
```

innan T1 är klar med sina beräkning. level kommer således att få värdet 150.
När sedan T1 blir klar med sin beräkning har newLevel värdet 50. T1 utför sedan satsen

```
level = newLevel;
```

och level får värdet 50. Men ett korrekt värde borde varit 100. Ett inkonsistent tillstånd har inträffat.

ii)

```
public class WaterTank {
   private double level;
   public synchronized void fill(double volume) {
      level = level + volume;
   }
   public synchronized boolean empty(double volume) {
      if (volume < level) {
         double newLevel = level - volume;
         level = newLevel;
         return true;
      }
      else
         return false;
   }
}//WaterTank
```

Uppgift 11.

a)

```
public AppStore() {
    store = new HashMap<String, Set<String>>();
}
```

b)

```
public void addApp(String osType, String appName) {
    Set<String> osTypeApps = store.get(osType);
    if (osTypeApps == null) {
        osTypeApps = new TreeSet<String>();
        store.put(osType, osTypeApps);
    }
    osTypeApps.add(appName);
}
```

c)

```
public String getOsTypeWithMaxNumApps() {
    int max = -1, numApps;
    String osTypeMaxApp = null;
    for (String osType : store.keySet()) {
        numApps = store.get(osType).size();
        if (numApps > max) {
            max = numApps;
            osTypeMaxApp = osType;
        }
    }
    return osTypeMaxApp;
}
```