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

DAG: 10-12-13 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 17/1 kl 10-12 och torsdag 20/1 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 papperet 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.

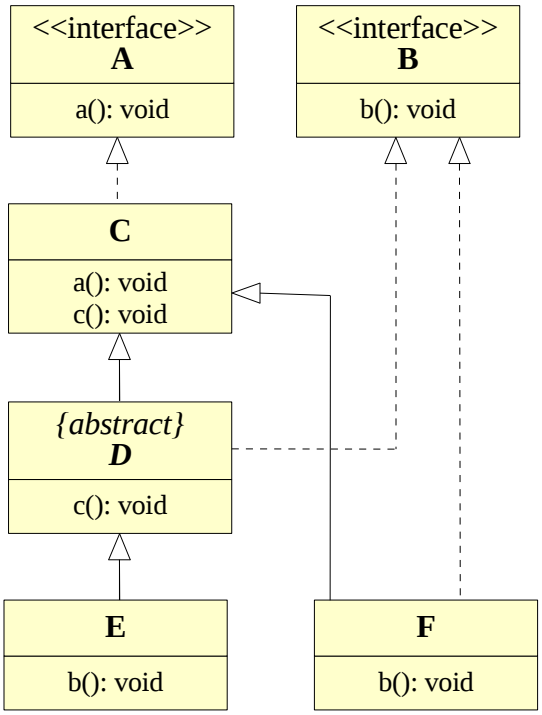
Vid rättning av uppgifter där programkod ingår bedöms principiella fel allvarligare än smärre språkfel.

LYCKA TILL!!!!

Uppgift 1.

Betrakta nedanstående klasser och interface:

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

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

public class C implements A {
    public void a() {
        System.out.println( "a() in C" );
    }//a
    public void c() {
        System.out.println( "c() in C" );
    }//c
}//C

public abstract class D extends C implements B {
    public void c() {
        System.out.println( "c() in D" );
    }//c
}//D

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

public class F extends C implements B {
    public void b() {
        System.out.println( "b() in F" );
    }//b
}//F
```



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

a)
```
C x = new E();
x.c();
```

b)
```
Object o = new E();
System.out.println(o instanceof D);
```

c)
```
C x = new D();
x.c();
```

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

e)
```
C x = new F();
x.b();
```

f)
```
B x = new F();
D y = (D) x;
x.b( );
```

g)
```
D x = new C();
x.a();
```

h)
```
A x = new F();
C y = x;
y.a();
```

(8 poäng)

Uppgift 2.

a) Betrakta nedanstående två klasser:

```
public abstract class Payment {
    /**
      @pre  amount >= 500
    **/
    public void setPaymentAmount(int amount) { . . . }
    . . .
}//Payment

public class CreditCardPayment extends Payment {
    /**
      @pre  amount >= 1000
    **/
    public void setPaymentAmount(int amount) { . . . }
    . . .
}// CreditCardPayment
```

Överskuggar klassen `CreditCardPayment` metoden `setPaymentAmount` på ett korrekt sätt? Motivera ditt svar!

b) Betrakta nedanstående två klasser:

```
public class Pump {
    /**
      @post value returned is > 0
    **/
    public double volumePumped() { . . . }
    . . .
}//Pump

public class PropanePump extends Pump {
    /**
      @post  value returned is > 0 and divisible with 5
    **/
    public double volumePumped() { . . . }
    . . .
}//  PropanePump
```

Överskuggar klassen `PropanePump` metoden `volumePumped` på ett korrekt sätt? Motivera ditt svar!

c) Betrakta nedanstående två specifikationer av metoden `mean`:

```
/**
  @pre  arr ! = null and arr has at least one element.
  @return Returns the mean of the numbers in arr
**/
public static double mean(int[] arr) { . . . }

/**
  @pre  none
  @return If arr is null or empty throws IllegalArgumentException. Otherwise, the mean of the numbers in arr
**/
public static double mean(int[] arr) { . . . }
```

Vilken av specifikationerna är att föredra? Motivera ditt svar!

(6 poäng)

Uppgift 3.

a) I kodsegmentet nedan gäller att klassen `DomsException` är en subklass till `Exception`. Kodsegmentet går inte att kompilera. Förklara varför och korrigera koden.

```
try {
    throw new DomsException();
} catch (Exception e) {
    System.out.println("Caught an exception!");
} catch (DomsException e) {
    System.out.println("Caught Dom's exception!");
} finally {
    System.out.println("This is the finally block!");
}
```

b) Betrakta nedanstående klasser:

```
public class SuperClass {
    private int value;
    private String description;
    public SuperClass(int val, String desc) {
        value = val;
        description = desc;
    }//constructor
    public String toString() {
        return (description + " has value " + value);
    }//toString
}//SuperClass

public class SubClass extends SuperClass {
    private int secondValue;
    public SubClass(int val, String desc, int val2) {
        description = desc;
        value = val;
        secondValue = val2;
    }//constructor
    public String toString(){
        return (description + " has value " + value + " & second value " + secondValue);
    }//toString
}//SubClass
```

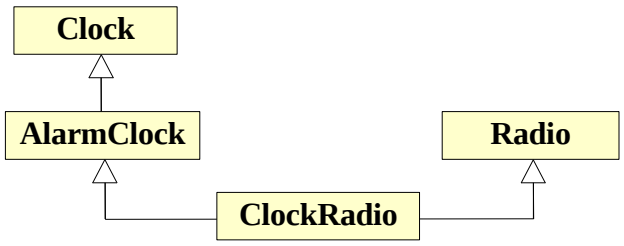
Klassen `SubClass` går inte att kompilera. Gör de ändringar i klassen `SubClass` som behövs för att den skall gå att kompilera. Klassen `SuperClass` får inte förändras.

(4 poäng)

Uppgift 4.

Klassdiagrammet bredvid uttrycker förhållandena som gäller mellan klasserna `ClockRadio`, `Radio`, `AlarmClock` och `Clock`.

Denna design går inte att direkt implementera i Java.



Beskriv vad problemet består i, samt ge ett nytt förslag på en design som löser problemet. Du kan ge din lösning antingen som ett UML-diagram eller som en uppsättning kodskelett.

(6 poäng)

Uppgift 5.

I ett övervakningssystem för broar, som är särskilt utsatta för vind, använder man en abstrakt klass `Vehicle` för att representera fordon. Klassen har flera subklasser, bl.a. `Car`, `Lorry`, `Motorcycle` och `MobileHome`. Eftersom de två sistnämnda fordonstyperna är särskilt vindkänsliga har man implementerat en metod removeWindSensitiveVehicles som kan ta bort sådana ur en fordonslista.

```
public class VehicleUtil {
    // omissions
    public static void removeWindSensitiveVehicles(List<Vehicle> list) {
        Iterator<Vehicle> iterator = list.iterator();
        while(iterator.hasNext()) {
            Vehicle vehicle = iterator.next();
            if( vehicle instanceof Motorcycle || vehicle instanceof MobileHome) {
                iterator.remove();
            }
        }
    }// removeWindSensitiveVehicles
}// VehicleUtil
```

Modifiera designen så att man inte behöver ändra metoden om det skulle tillkomma fler vindkänsliga fordonstyper. Lösningen redovisas med de förändringar som måste göras i de existerande fordonsklasserna och i metoden `removeWindSensitiveVehicles` i klassen `VehicleUtil`. Om du väljer att införa ytterligare programenheter skall dessa också redovisas.

(8 poäng)

Uppgift 6.

En stack är en datastruktur som har ett LIFO-beteende, dvs det sista elementet som läggs på stacken är det första elementet som tas bort från stacken. Följande interface är givet för en stack:

```
public interface Stack<E> {
    // returns tar bort och returnerar det översta elementet i stacken
    // kastar EmptyStackException om stacken är tom
    E pop() throws EmptyStackException ;

    // returnerar det översta elementet i stacken
    //kastar EmptyStackException om stacken är tom
    E top() throws EmptyStackException;

    //lägger in elementet e överst på stacken
    void push(E e);

    //returnerar true om stacken är tom, annars returneras false
    boolean isEmpty();
}//Stack
```

Skriv en implementation `ListStack` av interfacet `Stack`. I implementationen `ListStack` skall elementen i stacken internt lagras i en `ArrayList` eller i en `LinkedList`. Motivera ditt val.

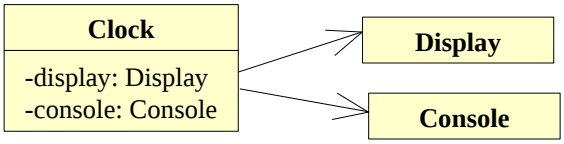
Klassen `EmptyStackException` finns deklarerad i paketet ‚`java.util`.

(8 poäng)

Uppgift 7.

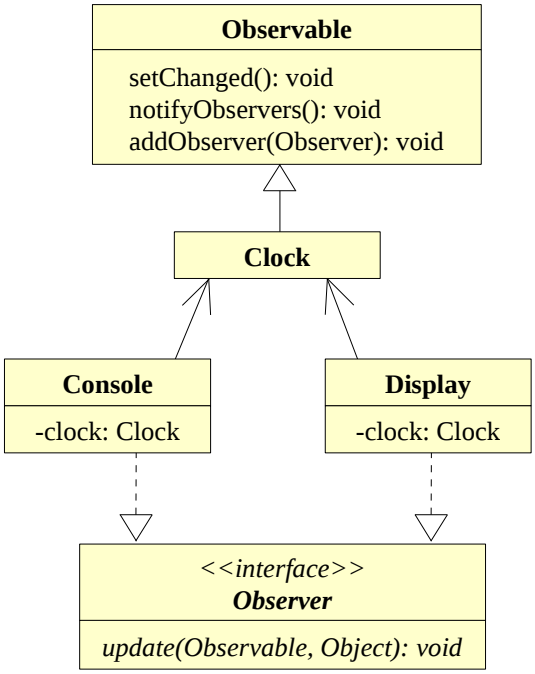
Nedan har vi de tre klasserna `Clock`, `Display` och `Console`. Klassen `Clock` används för att avbilda en klocka och innehåller bl.a. metoden `tick` som räknar fram tiden. Klasserna `Display` och `Console` har båda metoder för att visa aktuell tid, metoderna `show` respektive `print`.

```
public class Clock {
    private Time time;
    private Display display;
    private Console console;
    public Clock(Display display, Console console) {
        this.display = display;
        this.console = console;
    }//constructor
    public void tick() {
        time.increase();
        display.show(time);
        console.print(time);
    }//tick
    ...
}//Clock
public class Display {
    public void show(Time time) { ... }
    ...
}//Display
public class Console {
    public void print(Time time) { ... }
    ...
}//Console
```



Designen har bristen att `Clock` är beroende av klasserna `Display` och `Console`. Det borde vara tvärtom.

Ändra designen så att designmönstret `Observer` realiseras (enligt UML-diagramet nedan). Det räcker om du visar koden för klasserna `Clock` och `Display` (`Console` blir analog med `Display`).



(8 poäng)

Uppgift 8.

I ett Java-program som används för att testa stresståligheten hos en grupp försökspersoner, går ett experimentet ut på att försökspersonen skall göra en förutbestämd följd av knapptryckningar. Om en knapptryckning blir felaktig startas en tråd av klassen `ShutdownThread` (se nedan) som skriver ut ett felmeddelande, varefter tråden lägger sig och sover en viss tid, under vilken försökspersonen får chansen att korrigera sitt fel (genom att ge en ny sekvens av knapptryckningar). Lyckas försökspersonen med detta inom tiden tråden sover, avbryts tråden med ett anrop av `interrupt()`, annars skjuter tråden ner programmen genom att anropa `System.exit(0)`.

Din uppgift är att skriva klassen `ShutdownThread`. Klassen `ShutdownThread` skall ha har en konstruktor

public ShutdownThread(String msg, **int** seconds)

där parametern `msg` anger felmeddelandet som skrivs ut och parametern `seconds` anger hur länge försökspersonen har på sig att rätta till sitt fel innan systemet skjuts ner.

Tips: Om tråden blir avbruten görs lämpligen **return**, för att avsluta tråden på ett kontrollerat sätt.

(8 poäng)

Uppgift 9.

I ett Java-program som handhar en stegtävling mellan olika lag finns nedanstående klass som deltagarna i ett lag använder för att rapportera sina resultat:

```
public class Team {
    private int totalSteps = 0;
    public void report(int steps) {
        totalSteps = totalSteps + steps;
    }//report
    public int read() {
        return totalSteps;
    }//read
}//Team
```

Problemet är att klassen inte är trådsäker.

a) Beskriv med ett exempel vad som kan inträffa.

b) Gör klassen trådsäker.

(4 poäng)

Tentamen 101213 - LÖSNINGSFÖRSLAG

Uppgift 1.

a) Ger utskriften ”c( ) in D”

b) Ger utskriften ”true”

c) Tilldelningen C x = **new** D() ger kompileringsfel eftersom klassen D är abstrakt.

d) Ger utskriften ”b( ) in E”

e) Anrope x.b() ger kompileringsfel eftersom typen C inte har operationen b().

f) Ger exekveringsfel. Typen F kan inte typomvandlas till typen D.

g) Tilldelningen D x = **new** C() ger kompileringfel, eftersom typen C inte är subtyp till D.

h) Tilldelningen C y = x ger kompileringsfel, eftersom x är av typen A och A är ingen subtyp till C.

Uppgift 2.

a) Nej! En subklass får inte ha ett starkare förvillkor på en överskuggad metod än vad superklassen har.

b) Ja! En subklass får ha ett starkare eftervillkor på en överskuggad metod än vad superklassen har.

c) Den andra specifikationen

```
/**
  @pre  none
  @return If arr is null or empty throws IllegalArgumentException. Otherwise, the mean of the numbers in arr
**/
public static double mean(int[] arr) { . . . }
```

är att föredra, eftersom denna metod inte har några förvillkor som användaren måste uppfylla för att kunna använda metoden.

Uppgift 3.

a) Satsen

```
catch (DomsException e)
```

kommer aldrig kan nås eftersom DomsException är en subklass till Exception och därmed fångas undantag av typen DomsException av satsen

```
catch (Exception e)
```

Koden skall ha följande utseende:

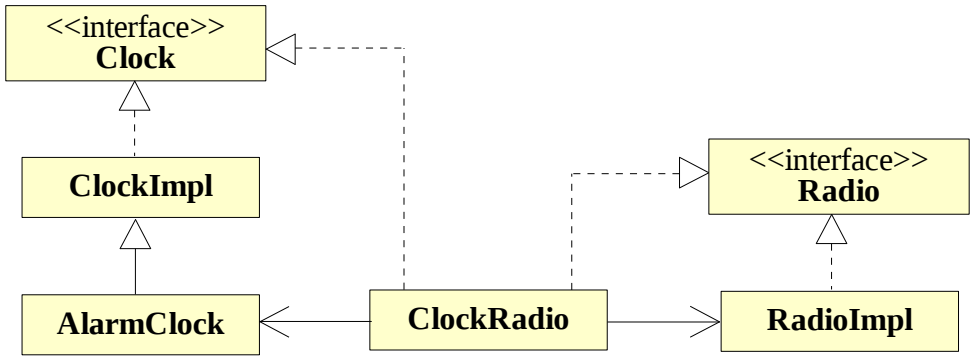
```
try {
    throw new DomsException();
} catch (DomsException e) {
    System.out.println("Caught Dom's exception!");
} catch (Exception e) {
    System.out.println("Caught an exception!");
} finally {
    System.out.println("This is the finally block!");
}
```

b) `SubClass` kan inte referera till privata instansvariabler i sin superklass `SubClass`. En korrekt lösning har följande utseende:

```
public class SubClass extends SuperClass {
    private int secondValue;
    public SubClass(int val, String desc, int val2) {
        super(val, desc);
        secondValue = val2;
    }//constructor
    public String toString(){
        return (super.toString() + " & second value " + secondValue);
    }//toString
}//SubClass
```

Uppgift 4.

Lösning presenterad som UML-diagram:



Lösning som kodskelett:

```
public interface Clock {
    . . .
}//Clock

public class ClockImpl implements Clock {
    . . .
}// ClockImpl

public class AlarmClock extends ClockImpl {
    . . .
}// AlarmClock

public interface Radio {
    . . .
}//Radio

public class RadioImpl implements Radio {
    . . .
}// RadioImpl

public class ClockRadio implements Clock, Radio {
    private Clock clock = new AlarmClock();
    private Radio radio = new RadioImpl();
    . . .
}
```

Uppgift 5.

```
public abstract class Vehicle {
    . . .
    public boolean isSensitive();
}// Vehicle

public class Car extends Vehicle {
    . . .
    public boolean isSensitive() {
        return false;
    }// isSensitve
}//Car

public class Lorry extends Vehicle {
    . . .
    public boolean isSensitive() {
        return false;
    }// isSensitve
}//Lorry

public class Motorcycle extends Vehicle {
    . . .
    public boolean isSensitive() {
        return true;
    }// isSensitive
}// Motorcycle

public class MobileHome extends Vehicle {
    . . .
    public boolean isSensitive() {
        return true;
    }// isSensitve
}// MobileHome

public class VehicleUtil {
    // omissions
    public static void removeWindSensitiveVehicles(List<Vehicle> list) {
        Iterator<Vehicle> iterator = list.iterator();
        while(iterator.hasNext()) {
            Vehicle vehicle = iterator.next();
            if(vehicle.isSensitive ()) {
                iterator.remove();
            }
        }
    }// removeWindSensitiveVehicles
}// VehicleUtil
```

Man skulle också kunna införa ett interface

```
public interface Sensitive {
    public boolean isSensitive();
}// Sensitive
```

som klassen `Vehicle` implementerar

```
public abstract class Vehicle implements Sensitive {
    . . .
}// Vehicle
```

I övrigt blir klasserna som ovan.

Även andra lösningar är möjliga.

Uppgift 6.

Att använda LinkedList i implementationen är att föredra då implementationen blir enklare och möjligen också mera effektiv.

Implementation med LinkedList:

```
import java.util.*;
public class ListStack<E> implements Stack<E> {
    private LinkedList<E> elements;
    public ListStack() {
        elements = new LinkedList<E>();
    }//constructor
    public void push(E e) {
        elements.addFirst(e);
    }//push
    public E top() {
        if (elements.isEmpty())
            throw new EmptyStackException();
        return elements.getFirst();
    }//top
    public E pop() {
        if (elements.isEmpty())
            throw new EmptyStackException();
        return elements.removeFirst();
    }//pop
    public boolean isEmpty() {
        return elements.isEmpty();
    }//isEmpty
}//ListStack
```

Implementation med ArrayList:

```
import java.util.*;
public class ListStack<E> implements Stack<E> {
    private List<E> elements;
    public ListStack() {
        elements = new ArrayList<E>();
    }//constructor
    public void push(E e) {
        elements.add(e);
    }//push
    public E top() {
        if (elements.isEmpty())
            throw new EmptyStackException();
        return elements.get(elements.size()-1);
    }//top
    public E pop() {
        if (elements.isEmpty())
            throw new EmptyStackException();
        return elements.remove(elements.size()-1);
    }//pop
    public boolean isEmpty() {
        return elements.isEmpty();
    }//isEmpty
}//ListStack
```

Uppgift 7.

```
import java.util.Observable;
public class Clock extends Observable {
    private Time time;
    public void tick() {
        time.increase();
        setChanged();
        notifyObservers();
    }//constructor
    public Time getTime() {
        return time;
    }//getTime
}//Clock

import java.util.Observer;
import java.util.Observable;
public class Display implements Observer {
    private Clock clock;
    public Display(Clock clock) {
        this.clock = clock;
        clock.addObserver( this );
    }//constructor
    private void show(Time time) {
        ...
    }//show
    public void update(Observable obs, Object o) {
        show(clock.getTime());
    }//update
}//Display
```

Behövde inte skrivas

```
import java. Observer;
import java. Observable;
public class Console implements Observer {
    private Clock clock;
    public Console(Clock clock) {
        this.clock = clock;
        clock.addObserver(this);
    }//constructor
    private void print(Time time) {
        ...
    }//print
    public void update(Observable obs, Object o) {
        print(clock.getTime());
    }//update
}//Console
```

Uppgift 8.

```
public class ShutdownThread extends Thread {
    private int secs;
    private String msg;

    public ShutdownThread(String msg, int secs) {
        this.secs = secs;
        this.msg = msg;
    }//constructor

    public void run() {
        System.out.println(msg);
        try {
            Thread.sleep(secs*1000);
        } catch (InterruptedException e) {
            return;
        }
        shutdownNow();
    }//run

    public static void shutdownNow() {
        System.out.println("You failed!");
        System.exit(0); // cause entire JVM to shut down
    }// shutdownNow
}// ShutdownThread
```

Uppgift 9.

a) Flera trådar kan samtidigt anropa metoden `report`. Effekten kan bli att ett resultat som rapporteras in inte kommer med i slutresultatet.

Exempel:

Anta att instansvariabeln totalStep har värdet 123456 när två trådar t1 och t2 samtidigt exekverar metoden report. I beräkningen av satsen

```
totalSteps = totalSteps + steps;
```

läser både t1 och t2 av värdet 123456 på instansvariabeln totalStep.

t1 adderar 1234 till värdet 123456 och får resultatet 124690.

t2 adderar 4321 till värdet 123456 och får resultatet 127777.

t2 lagrar värdet 127777 i instansvariabel `totalStep`.

t1 lagrar värdet 124690 i instansvariabeln `totalStep`.

Effekten blir att de 4321 stegen som t1 skulle registrera aldrig blev registrerade i slutresultatet.

b)

```
public class Team {
    private int totalSteps = 0;
    public synchronized  void report(int steps) {
        totalSteps = totalSteps + steps;
    }//report
    public int read() {
        return totalSteps;
    }//read
}//Team
```