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

DAG: 11-12-12 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 16/1 kl 12-13 och torsdag 19/1 kl 12-13, rum 6128 i EDIT-huset.
Hjälpmedel:	Inga hjälpmedel är tillåtna förutom bilagan till tesen.
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 principella fel allvarligare än smärre språkfel.

LYCKA TILL!!!!

Uppgift 1.

Betrakta nedanstående klasser:

```
public class A{
    private int n = 0;

    public A() {
        this.n = 0;
        System.out.println("A constructor1");
    }//constructor

    public A(int n){
        this.n = n;
        System.out.println("A constructor2");
    }//constructor

    public String toString(){
        return "" + n;
    }//toString
}//A
```

```
public class B extends A{
    private String text;

    public B(int n, String text){
        super(n);
        this.text = text;
        System.out.println("B constructor1");
    }//constructor

    public B(String text){
        this.text = text;
        System.out.println("B constructor2");
    }//constructor

    public String getText(){
        return text;
    }//getText
}//B
```

a. Vad blir utskriften när följande sats exekveras?

```
B b = new B("bbb");
```

b. Antag att b är deklarerade enligt deluppgift a). Är nedanstående sats giltig? Om svaret är ja, vad kommer att skrivas ut? Om svaret är nej, förklara vad som är fel.

```
System.out.println("b = " + b);
```

c. Är nedanstående satser giltiga? Om svaret är ja, vad blir utskriften av sista raden? Om svaret är nej, ange vilken rad som orsakar felet och förklara varför det blir fel.

```
1. A x1 = new A(5);
2. B x2 = new B(7, "bbb");
3. x1 = x2;
4. System.out.println(x1.getText());
```

d. Är nedanstående satser giltiga? Om svaret är ja, vad blir utskriften av sista raden? Om svaret är nej, ange vilken rad som orsakar felet och förklara varför det blir fel.

```
1. A x1 = new A();
2. B x2 = new B("bbb");
3. x2 = x1;
4. System.out.println(x2.getText());
```

e. Överskugga metoden toString i klassen B, på så sätt att metoden returnerar värdet av attributet n åtföljt av värdet på attributet text.

f. Antag att metoden toString i deluppgift e) har lagts till klassen B på ett korrekt sätt. Är nedanstående satser giltiga? Om svaret är ja, vad blir utskriften av sista raden? Om svaret är nej, ange vilken rad som orsakar felet och förklara varför det blir fel.

```
1. A x1 = new A(3);
2. B x2 = new B(4,"xxx");
3. x1 = x2;
4. System.out.println(x1.toString());
```

(6 poäng)

Uppgift 2.

a) Vad skall det finnas för relation (kontrakt) mellan equals() och hashCode()? (2 poäng)

b) Betrakta klassen House nedan:

```
public class House {
    private int windows;
    private int doors;
    private int chimneys;
    public House(int windows, int doors, int chimneys) {
        this.windows = windows;
        this.doors = doors;
        this.chimneys = chimneys;
    }
    public int hashCode() {
        return windows + doors;
    }
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        House other = (House) obj;
        return // <-- insert code here
    }
}
```

Vilka av nedanstående kodfragment kan läggas till för att komplettera metoden equals() utan att kontraktet till hashCode() bryts? Rätt svar ger 0.5 poäng och felaktigt svar ger -0.5 poäng.

I) windows == other.windows

II) windows == other.windows && doors == other.doors

III) windows == other.windows && doors == other.doors && chimneys == other.chimneys

IV) windows + doors == other.windows + other.doors

V) windows + doors + chimneys == other.windows + other.doors + other.chimneys

VI) windows == other.windows || doors == other.doors

(3 poäng)

Uppgift 3.

Betrakta nedanstående program:

```
public class Tester {
   public static void main(String[] args) {
      List<Dog> animals = new ArrayList<Dog>();
      animals.add(new Dog("Fido"));
      animals.add(new Dog("Harvey"));
      message(animals);
   }

   private static void message(List<Animal> animals) {
      System.out.println("You gave me a collection of animals.");
   }

   private static void message(Object object) {
      System.out.println("You gave me an object.");
   }
}

public interface  Animal {. . .}

public class Dog implements Animal {
    private String name;
    public Dog(String name) {
        this.name = name;
    }
    . . .
}
```

Vad skrivs ut av program? Förklara varför denna utskrift erhålls!

(3 poäng)

Uppgift 4.

Betrakta nedanstående klass:

```
public class NextNumber {
    private int number;
    public NextNumber() {
        this.number = 0;
    }
    public int getNext() {
        number++;
        return number;
    }
}
```

Skriv om klassen på så sätt att klassen implementerar designmönstret *Singleton*, och att klassen kan användas i en applikation med flera trådar.

(4 poäng)

Uppgift 5.

Betrakta nedanstående klass som är tagen ur ett program som modellerar en räknedosa.

```
public class Key extends JButton implements ActionListener {
    private Calculator calculator;
    private String operation;
    public Key(String operation, Calculator calculator) {
        super(operation);   //Creates a button with initial text.
        this.operation = operation;
        this.calculator = calculator;
        addActionListener(this);
    }//constructor
    public void actionPerformed(ActionEvent e) {
        if (operation.equals("+")) {
            calculator.add();
        } else if (operation.equals("-")) {
            calculator.sub();
        } else if (operation.equals("enter")) {
            calculator.enter();
        }
    }// actionPerformed
}//Key
```

Klassen Key används för att modellera och visa tangenter. I denna version finns endast tre operationstangenter: *add*, *sub* och *enter*. Designen strider mot *Open/Closed-principen*; man kan inte lägga till ytterligare tangenter utan att modifiera Key-klassen. Åtgärda detta genom att implementera en ny design som använder Strategy-mönstret.

(7 poäng)

Uppgift 6.

Betrakta nedanstående klass:

```
/** An immutable class for representing a point in n-dimensional space.
   @specfield dimension the number of dimensions of the space the point is in
   @specfield position the position in n-dimensional space of the point
**/
public class PointND {
    private List<Double> components;

    /** Creates a new PointND with dimension = the length of components and having the position corresponding
        to the ordered list: (components0, components1, ..., components(dimension-1) )
        @Requires  components != null
    **/
    public PointND(List<Double> components) {
        this.components = components;
    }

    /** @returns the dimension of this  **/
    public int getDimension() {
        return components.size();
    }
    //... more methods are omitted for the sake of brevity
}
```

Som framgår av specifikationen är avsikten att klassen skall vara icke-muterbar, men klassen är inte icke-muterbar.

a) Förklara varför!

b) Åtgärda! (4 poäng)

Uppgift 7.

Betrakta nedanstående specifikation:

```
public class Bicycle {
    ...
    /**
        requires: windspeed < 20mph && daylight
        effects: transports rider from work to home
    **/
    public void goHome () { ... }
}
```

a) Är klassen LightedBicycle nedan en *äkta* subtyp till Bicycle i enligt med *Liskov Substitution Principle*? Motivera ditt svar!

```
public class LightedBicycle extends Bicycle {
    ...
    /**
        requires: windspeed < 20mph
        effects: transports rider from work to home
    **/
    public void goHome () { ... }
}
```

b) Är klassen RacingBicycle nedan en *äkta* subtyp till Bicycle i enligt med *Liskov Substitution Principle*? Motivera ditt svar!

```
public class RacingBicycle extends Bicycle {
    ...
    /**
        requires: windspeed < 20mph && daylight
        effects: transports rider from work to home in elapsed time < 10 minutes
    **/
    public void goHome () { ... }
}
```

c) Är klassen TandemBicycle nedan en *äkta* subtyp till Bicycle i enligt med *Liskov Substitution Principle*? Motivera ditt svar!

```
public class TandemBicycle extends Bicycle {
    ...
    /**
        requires: windspeed < 20mph && daylight  && two riders
        effects: transports riders from work to home
    **/
    public void goHome () { ... }
}
```

(6 poäng)

Uppgift 8.

Betrakta nedanstående klass och interface:

```
public class Client {
    private ImperialSensor sensor;
    public Client(ImperialSensor sensor) {
        this.sensor = sensor;
    }
    public void calculate() {
        double distance = sensor.getDistance();
        double speed = sensor.getSpeed();
        . . .
    }
    . . .
}

public interface ImperialSensor {
   // returns: distance traveled in miles
   public double getDistance();

   // returns: speed in miles per hour
   public double getSpeed();
}
```

Klassen Client använder således en sensor av typen ImperialSensor, problemet är dock att den enda konkreta sensor som finns tillgänglig är av typen MetricSensor med följande utseende:

```
public class MetricSensor {
   // returns: distance traveled in meters
   public double readDistance() { . . . }

   // returns: speed in meters per second
   public double readSpeed() { . . . }
}
```

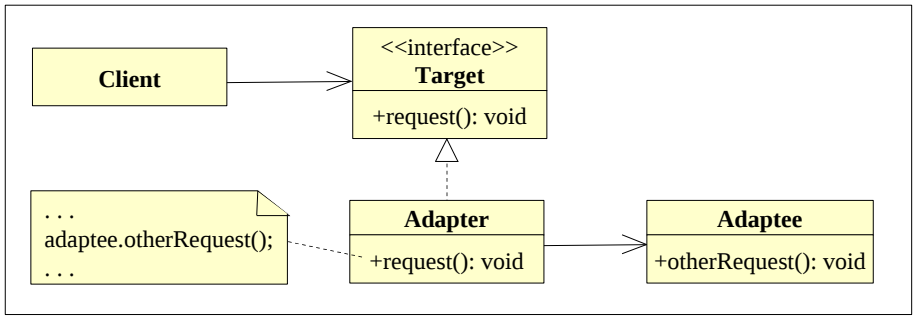
För att avhjälpa detta problem är din uppgift att skriva en klass MetricSensorToImperialSensor som implementerar *designmönstret Adapter* för att anpassa gränssnittet för typen MetricSensor till gränssnittet av typen ImperialSensor.

Tips: Följande konstanter kan behövas

```
MILES_PER_METER = 0.000621371192
HOURS_PER_SEC = 0.000277777778
```

(6 poäng)

Designmönstret Adapter har följande utseende:



Uppgift 9.

I ett Java-program som används vid en idrottstävling med individuella starttider används följande enkla klass för att representera resultat:

```
public class Competitor {
    private final String name;
    private double time;
    public Competitor(String name, double time) {
        this.name = name;
        this.time = time;
    }//constructor
    public double getTime() {
        return time;
    } //getTime
    public void setTime(double time) {
        this.time = time;
    } //setTime
    public String getName() {
        return name;
    } //getName
    public String toString(){
        return name + "\t" + time;
    }//toString
}// Competitor
```

När de tävlande går i mål räknas tiden ut och resultanten lagras i följande lista:

```
List<Competitor> results = new LinkedList<Competitor>();
```

När tävlingen är slut skrivs listan ut med följande satser:

```
for ( Competitor comp : results)
    System.out.println(comp);
```

varvid man erhåller en resultatlista där ordningen på de tävlande blir enligt den ordning de kom i mål (lades in i listan)

Moberg, Allan	31.34
Persson, Emil	34.12
Andersson, Knut	35.58
Öqvist, Olle	33.15
Nilsson, Frank	30.35
Cedervall, Rune	32.27
Lanz, Knut	29.04
Bertilsson, Bo	29.51
Ohlin, Fabian	29.14
Stenhuvud, Carl	29.23

De tävlande önskar dock även att få en resultatlista i placeringsordning enligt:

1.	Lanz, Knut	29.04
2.	Ohlin, Fabian	29.14
3.	Stenhuvud, Carl	29.23
4.	Bertilsson, Bo	29.51
5.	Nilsson, Frank	30.35
6.	Moberg, Allan	31.34
7.	Cedervall, Rune	32.27
8.	Öqvist, Olle	33.15
9.	Persson, Emil	34.12
10.	Andersson, Knut	35.58

Det är din uppgift att åstadkomma detta genom att implementera en metod

public static void printPlacement(List<Competitor> theList)

Du vet att klassen Collections innehåller en metod

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

och tänker utnyttja detta i din implementation. Således måste du också implementera en klass

public class TimeComparator **implements** Comparator<Competitor>

som realiserar metoden compare på lämpligt sätt.

(6 poäng)

Uppgift 10.

Betrakta klasserna Bee och Flower nedan

```
public class Bee {
    private final String name;
    public Bee(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    //. . .
}

public class Flower {
    private final String name;
    public Flower(String name) {
        this.name = name;
    }
    public void visit(Bee b) {
        System.out.println(name + " is being visited by " + b.getName());
    }
    public void bloom() {
        System.out.println(name + " is blooming!");
    }
    //...
}
```

Komplettera dessa klasser så att designmönstret Observer realiseras (genom att använda Observable och Observer eller genom att använda PropertyChangeSupport och PropertyChangeListener). När en blomma blommar meddelar den detta till alla bin som är observatörer, och när ett bi som får reda på att en blomma blommar besöker biet blomman. Nedanstående huvudprogram illustrerar hur det hela är tänkt att fungera:

```
public static void main(String[] args) {
    Flower f = new Flower("Tulip");
    Bee b1 = new Bee("Bob");
    Bee b2 = new Bee("Fred");
    f.addObserver(b1);
    f.addObserver(b2);
    f.bloom();
  }
```

Utskriften blir:

```
Tulip is blooming!
Tulip is being visited by Fred
Tulip is being visited by Bob
```

(5 poäng)

Uppgift 11.

Betrakta klassen Auction nedan som utgör en av komponenterna i ett system för att handha auktioner över nätet (du behöver inte förstå alla detaljerna för att kunna lösa uppgiften):

```
public class Auction extends Thread {
    private int currentBid;
    private String highestBidder = "Nobody";
    private boolean biddingOpen = true;
    public Auction (int minBid) {
        currentBid = minBid;
    }

    /* Return the current highest bid. */
    public synchronized int getCurrentBid () {
        return currentBid;
    }

    /* Return name of the bidder who has the highest bid. */
    public synchronized String getHighestBidder() {
        String bidder = highestBidder;
        return highestBidder;
    }

    /* Return true if the auction is going on, else false. */
    public synchronized boolean biddingAllowed () {
        return biddingOpen;
    }

    /* Close the auction.  */
    private  synchronized void closeBidding () {
        biddingOpen = false;
    }

    /* Will accept newBid from theBidder if the bid is higher than currently offered bid.  If the bid is accepted, */
    /* bidder thread blocks here until another bidder give a higher bid or the bidder wins the auction. */
    public synchronized void setBid (int newBid, String theBidder) {
        if (newBid > currentBid) {
            highestBidder = theBidder;
            currentBid = newBid;
            System.out.println("Bid of " + currentBid + " from " + theBidder + " accepted");
            notifyAll();
            try {
                while (biddingAllowed() && theBidder == highestBidder)
                    wait();
                }
            }
            catch (InterruptedException e) {};
        }
    }

    public void run() {
        while (biddingAllowed()) {
            int lastBid = currentBid;
            try {
                sleep(3000);
            } catch (InterruptedException e ){}
            if (lastBid == currentBid)
                closeBidding();
        }
        System.out.println("Sold to " + highestBidder);
    }
}
```

currentBid holds the current highest bid.

highestBidder holds the name of the bidder who has the highest bid.

biddingOpen has the value true if the auction is going on, else the value false.

Lowest bid accepted must be higher than minBid.

Checks each three seconds to see if the current highest bid has been raised.
If not the auction ends and the goods goes to last highest bidder.

För att få till stånd en auktion måste det också finnas budgivare. Din uppgift är att implementera klass

public class Bidder **extends** Thread

som modellerar budgivarna i auktionen.

I nedanstående main-metod framgår hur en auktion med tre budgivare skapas.

```
public class Main {
    public static void main (String [] args) {
        Auction aution = new Auction(110);
        Bidder bidder1 = new Bidder ("Bidder 1", aution, 170);
        Bidder bidder2 = new Bidder ("Bidder 2", aution, 120);
        Bidder bidder3 = new Bidder ("Bidder 3", aution, 150);
        aution.start();
        bidder1.start();
        bidder2.start();
        bidder3.start();
    }
}//Main
```

Exempel på utskrift:

```
Bid of 111 from Bidder 1 accepted
Bid of 112 from Bidder 2 accepted
. . .
Bid of 148 from Bidder 1 accepted
Bid of 149 from Bidder 3 accepted
Bid of 150 from Bidder 1 accepted
Sold to Bidder 1
```

Klassen Bidder har en konstruktor med tre argument - budgivarens namn, vilken auktion budgivaren skall delta i samt hur högt bud budgivaren är villig att ge. En budgivare deltar i auktionen så länge som det lagda budet ligger lägre än vad budgivaren själv är villig att ge eller så länge som auktionen pågår. I det senare fallet har budgivaren vunnit budgivningen. En budgivare höjer alltid det lagda budet med 1. I din implementation av Bidder skall du låta tråden sova en slumpmässig tid, om max en sekund, mellan varje budgivning.

(8 poäng)

Tentamen 111212- LÖSNINGSFÖRSLAG

Uppgift 1.

a) A constructor1
B constructor2

b) Satsen är giltig. Utskriften blir: b = 0

c) Kodsegmentet är ogiltigt. Rad 4 orsakar ett kompileringsfel, eftersom en superklass inte kan anropa en metod som deklareras i en subklass.

d) Koden är felaktig. Rad 3 orsakar ett kompileringsfel, eftersom en referens av typ B inte kan referera till ett objekt av typ A.

e)
```
public String toString(){
    return super.toString() + text;
}
```

f) Koden är giltig. Utskriften blir: 4xxx

Uppgift 2.

a) Det skall gälla att

om x.equals(y), så är x.hashCode() == y.hashCode()

b)

I) windows == other.windows
Felaktig!

II) windows == other.windows && doors == other.doors
Korrekt!

III) windows ==other.windows && doors == other.doors && chimneys == other.chimneys
Korrekt!

IV) windows + doors == other.windows + other.doors
Korrekt!

V) windows + doors + chimneys == other.windows + other.doors + other.chimneys
Felaktig

VI) windows == other.windows || doors == other.doors
Felaktig

Uppgift 3.

Utskriften blir:

You gave me an object.

Orsaken är att List<Dog> inte är en subtyp till List<Animal>.

Tänk för en stund att List<Dog> vore en subtyp till List<Animal>, så betyder det att List<Dog> är utbytbar mot List<Animal> . Om vi då har en klass

class Cat **extends** Animal

är det naturligtvis tillåtet att lägga in ett objekt av klassen Cat i en List<Animal> och då är det också tillåtet att lägga in ett objekt av klassen Cat i List<Dog> eftersom List<Dog> är utbytbar mot List<Animal>.

Men en katt är inte en hund! Därför är inte heller List<Dog> en subtyp till List<Animal>!

Uppgift 4.

```
public class NextNumber {
    private static NextNumber instance;
    private int number;
    private NextNumber () {
        number = 0;
    }
    public static synchronized NextNumber getInstance() {
        if (instance == null)
            instance = new NextNumber();
        return instance;
    }
    public synchronized int getNext() {
        number++;
        return number;
    }
}
```

Uppgift 5.

```
public interface Operation {
    void execute(Calculator calculator);
    String getName();
}//Operation

public class Add implements Operation {
    public void execute(Calculator calculator) {
        calculator.add();
    }//execute
    public String getName() {
        return "+";
    }// getName
}//Add

public class Sub implements Operation {
    public void execute(Calculator calculator) {
        calculator.sub();
    }//execute
    public String getName() {
        return "-";
    }// getName
}//Sub

public class Enter implements Operation {
    public void execute(Calculator calculator) {
        calculator.enter();
    }//execute
    public String getName() {
        return "enter";
    }// getName
}//Equals

public class Key extends JButton implements ActionListener {
    private Calculator calculator;
    private Operation operation;
    public Key(Operation operation, Calculator calculator) {
        super(operation.getName ());
        this.operation = operation;
        this.calculator = calculator;
        addActionListener(this);
    }//constructor
    public void actionPerformed(ActionEvent e) {
            operation.execute(calculator);
    }// actionPerformed
}//Key
```

Uppgift 6.

a) Konstruktorn exponerar den interna representationen. Eftersom klienten som skapar ett objekt har en referens till listan components kan klienten när som helst ändra elementen i listan components bakom ryggen på PointND-objektet.

b) Konstruktorn skall inte lagra listan som ges som argument utan i stället en kopia av denna.

```
public PointND(List<Double> components) {
    this.components = new ArrayList<Double>(components);
}
```

Det var ett fel i bifogad API-utdrag där det angavs att List<E> har metoden clone(), vilket inte är sant. Därför godkänns även (den felaktiga) lösningen:

```
public PointND(List<Double> components) {
    this.components = (List<Double>) components.clone();
}
```

Uppgift 7.

a) LightedBicycle är en *äkta* subtyp till Bicycle eftersom en subtyp får ha ett svagare förvillkor

b) RacingBicycle är en *äkta* subtyp till Bicycle eftersom en subtyp får ha ett starkare eftervillkor.

c) TandemBicycle är inte en *äkta* subtyp till Bicycle eftersom en subtyp inte får ha ett starkare förvillkor.

Uppgift 8.

```
public class MetricSensorToImperialSensor implements ImperialSensor {
    public final static double MILES_PER_METER = 0.000621371192;
    public final static double HOURS_PER_SEC = 0.000277777778;
    private final MetricSensor ms;

  public MetricToSensorToImperialSensor() {
     ms = new MetricSensor();
  }

  // returns: distance traveled in miles
  public double getDistance() {
     return ms.readDistance() * MILES_PER_METER;
  }

  // returns: speed in miles per hour
  public double getSpeed() {
     return ms.readSpeed() * MILES_PER_METER / HOURS_PER_SEC;
  }
}
```

Uppgift 9.

Klassen TimeComparator:

```
import java.util.Comparator;
public class TimeComparator implements Comparator<Competitor> {
    public int compare( Competitor comp1, Competitor comp2) {
        if (comp1.getTime() < comp2.getTime())
            return -1;
        else if (comp1.getTime() > comp2.getTime())
            return 1;
        else
            return 0;
    }//compare
}//TimeComparator
```

Eller betydligt elegantare:

```
import java.util.Comparator;
public class TimeComparator implements Comparator<Competitor> {
    public int compare( Competitor comp1, Competitor comp2) {
        return (int) (comp1.getTime() - comp2.getTime());
    }//compare
}//TimeComparator
```

Metoden printPlacement:

```
public static void printPlacement(List<Competitor> theList) {
    List<Competitor> newList = new LinkedList<Competitor>(theList);
    Collections.sort(newList, new TimeComparator());
    int place = 0;
    for (Competitor comp : newList) {
        System.out.println(++place + "\t" + comp);
    }
}//printPlacement
```

Uppgift 10.

Lösning med Observer och Observeable:

```
import java.util.Observer;
import java.util.Observable;
public class Bee implements Observer {
    private final String name;
    public Bee(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void update(Observable obs, Object obj) {
        if (obs instanceof Flower) {
            Flower f = (Flower) obs;
            f.visit(this);
        }
    }
}
```

```
import java.util.Observable;
public class Flower extends Observable {
private final String name;
    public Flower(String name) {
        this.name = name;
    }
    public void visit(Bee b) {
        System.out.println(name + " is being visited by " + b.getName());
    }
    public void bloom() {
        System.out.println(name + " is blooming!");
        setChanged();
        notifyObservers();
    }
}
```

Lösning med PropertyChangeSupport och PropertyChangeListener:

Om inga förändringar skall göras i main-metoden i uppgiften måste vi införa ett interface:

```
import java.beans.PropertyChangeListener;
public interface IObservable {
    void addObserver(PropertyChangeListener observer);
    void removeObserver(PropertyChangeListener observer);
}

import java.beans.PropertyChangeSupport;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeEvent;
public class Flower implements IObservable {
    private PropertyChangeSupport pcs = new PropertyChangeSupport(this);
    private final String name;
    public Flower(String name) {
        this.name = name;
    }
    public void addObserver(PropertyChangeListener observer) {
        pcs.addPropertyChangeListener(observer);
    }
    public void removeObserver(PropertyChangeListener observer) {
        pcs.removePropertyChangeListener(observer);
    }
    public void visit(Bee b) {
        System.out.println(name + " is being visited by " + b.getName());
    }
    public void bloom() {
        System.out.println(name + " is blooming!");
        pcs.firePropertyChange("Some text", this, name); // eller några andra lämpliga parametrar
    }
}

import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeEvent;
public class Bee implements PropertyChangeListener {
    private String name;
    public Bee(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void propertyChange(PropertyChangeEvent ev) {
        if (ev.getSource() instanceof Flower) {
            Flower f = (Flower) ev.getSource();
            f.visit(this);
        }
    }
}
```

Uppgift 11.

```
public class Bidder extends Thread {
    private String bidderName;
    private Auction auction;
    private int maxBid;
    public Bidder (String name, Auction a, int top) {
        bidderName = name;
        auction = a;
        maxBid = top;
    }//constructor
    public void run () {
        while (auction.biddingAllowed()) {
            int currentBid = auction.getCurrentBid();
            if (currentBid < maxBid) {
                auction.setBid(currentBid + 1,bidderName);
                try {
                    sleep((int) (Math.random()*1000));
                } catch (InterruptedException e) {}
            }
            else
                break;
        }
    }//run
}//Bidder
```