Lösningsförslag till tentamen *P r e l i m i n ä r*

Kurs	**Objektorienterad programmering**
Tentamensdatum	**2012-10-25**
Program	**D2**
Läsår	**2012/2013, lp 1**
Examinator	**Uno Holmer**

Uppgift 1 (10 p)

Ingen lösning ges.

Uppgift 2 (10 p)

```
public class TimerButton extends JButton implements ActionListener {
    private static final int DELAY = 1000;
    private int setTime,time;
    private Timer timer;
    private boolean blinkOn;

    public TimerButton(int time) {
        setTime = this.time = time;
        blinkOn = true;
        setText("" + setTime);
        timer = new Timer(DELAY,this);
        addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if ( e.getSource() == timer )
            handleTick();
        else if ( e.getSource() == this )
            handlePush();
    }

    private void handleTick() {
        time--;
        if ( time <= 0 ) {
            if ( time == 0 ) {
                timer.setInitialDelay(200);
                timer.setDelay(200);
                timer.restart();
                setEnabled(true);
                blinkOn = true;
            }
            setText(blinkOn ? "ALARM!" : "");
            blinkOn = ! blinkOn;
        } else
            setText("" + time);
    }

    private void handlePush() {
        if ( time <= 0 ) {
            timer.stop();
            time = setTime;
```

```
            setText("" + setTime);
        } else {
            timer.setDelay(DELAY);
            timer.start();
            setEnabled(false);
        }
    }
}
```

Uppgift 3 (3+1+2+4 p)

a)

```
public final boolean equals(Object o) {
    if ( this == o )
        return true;
    else if (o instanceof Point) {
        Point other = (Point)o;
        return other.x == x && other.y == y;
    }
    return false;
}
```

b)

```
public int hashCode() {
    return 12345*x*37+y;
}
```

c)

```
public Point clone() {
    try {
        return (Point)super.clone();
    }
    catch (CloneNotSupportedException e) {
        throw new InternalError();
    }
}
```

d)

```
public Polygon clone() {
    try {
        Polygon copy = (Polygon)super.clone();
        copy.points = (ArrayList<Point>)points.clone();
        for ( int i = 0; i < points.size(); i++ )
            copy.points.set(i,points.get(i).clone());
        return copy;
    }
    catch (CloneNotSupportedException e) {
        throw new InternalError();
    }
}
```

Uppgift 4 (10 p)

```
public class TrafficData {
    private Map<Integer,long[]> table;

    public TrafficData(String fileName) throws IOException {
        table = new HashMap<Integer,long[]>();
        loadTable(fileName);
    }

    public boolean hasStation(int stationId) {
        return table.containsKey(stationId);
    }

    public long getNoOfVehicles(int stationId,int hour)
           throws IllegalArgumentException {
        if ( hour < 1 || hour > 24 )
            throw new IllegalArgumentException("TrafficData.getData:
                                                illegal hour");
        return table.get(stationId)[hour];
    }

    private void loadTable(String fileName) throws IOException {
        DataInputStream in =
            new DataInputStream(new FileInputStream(fileName));
        int stationId,hour;
        long noOfVehicles;
        while ( true ) {
            try {
                stationId = in.readInt();
                hour = in.readInt();
                noOfVehicles = in.readLong();
            }
            catch (EOFException e) {
                break;
            }
            long[] stationData = table.get(stationId);
            if ( stationData == null ) {
                stationData = new long[25];   // use 1..24
                table.put(stationId,stationData);
            }
            stationData[hour] = noOfVehicles;
        }
        in.close();
    }
}
```

Uppgift 5 (6+2+2 p)

a)

```
public class HammingGenerator extends NumberGenerator {
    private long currentHam = 0L;
    PriorityQueue<Long> queue = new PriorityQueue<Long>();

    public HammingGenerator() {
        reset();
    }
```

```
    public void computeNext() {
        currentHam = queue.poll();
        while ( ! queue.isEmpty() && queue.peek() == currentHam )
            currentHam = queue.poll();
        queue.add(2*currentHam);
        queue.add(3*currentHam);
        queue.add(5*currentHam);
        setChanged();
        notifyObservers("" + currentHam);
    }

    public void reset() {
        queue.clear();
        queue.add(1L);
        computeNext();
        setChanged();
        notifyObservers("" + currentHam);
    }
}
```

b)

- Lägg till inparametern `HammingGenerator hammingGen` till konstruktorn.
- Lägg till en panel i fönstret för Hammingtalen

```
JPanel hammingPanel =
    new NumberPane("Next Hamming number",
                   new NextButtonController(hammingGen),
                   new ResetButtonController(hammingGen));
hammingGen.addObserver((Observer)hammingPanel);
add(hammingPanel);
```

c)

- Lägg till/ändra i main:

```
HammingGenerator hammingGen = new HammingGenerator();
new UserInterface(primeGen,fiboGen,hammingGen);
hammingGen.reset();
```

Uppgift 6 (10 p)

```
public class SeekIterator<T> implements Iterator<T> {
    private Iterator<T> it;

    public SeekIterator(Iterator<T> it) {
        this.it = it;
    }

    public boolean hasNext() { return it.hasNext(); }
    public T next() { return it.next(); }
    public void remove() { it.remove(); }

    T seek(Property<T> p) {
        while ( it.hasNext() ) {
            T elem = it.next();
            if ( p.hasProperty(elem) )
                return elem;
        }
        return null;
    }
}
```