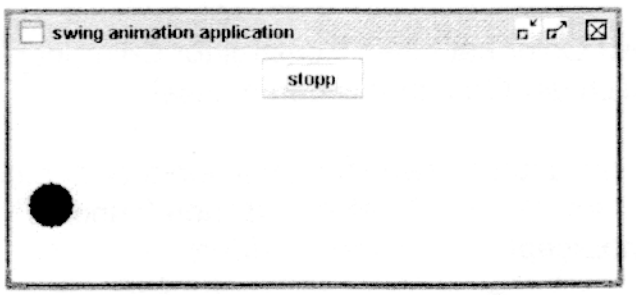
Name, Vorname: Matr.Nr.:

3. Versuch (Ja / Nein): Punkte: 62 Note 1.3 22.01.2005

1) (max. 20 Punkte)

Das folgende Programm zeichnet unter Verwendung von Swing einen blauen Kreis in ein Fenster.



Erweitern Sie diese Applikation so, dass der Kreis sich aus seiner Anfangsposition kontinuierlich horizontal nach rechts bewegt. Verwenden Sie dazu einen Thread (keinen Timer).

```
// Animation Application

import java.awt.*;
import javax.swing.*;

class MyFrame extends JFrame {

    private JButton button;
    private JPanel panel1;
    private MyPaintPanel panel2;

    public MyFrame(String str) {

        super(str);
        panel1 = new JPanel();
        panel2 = new MyPaintPanel();

        button = new JButton("stopp");

        Container c = getContentPane();
        panel1.setLayout(new FlowLayout());
        panel1.add(button);
        c.setLayout(new GridLayout(2,1));
        c.add(panel1);
        c.add(panel2);
    }
}

class MyPaintPanel extends JPanel {

    private int xPos = 10;
    private int yPos = 10;

    public MyPaintPanel() {
        super();
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.blue);
        g.fillOval(xPos,yPos,30,30);
    }
}

public class AnimationApplication {

   private static void createAndShowGUI() {

      final int WIDTH = 400;
      final int HEIGHT =180;

      JFrame.setDefaultLookAndFeelDecorated(true);
      MyFrame frame = new MyFrame("...");
      frame.setDefaultCloseOperation(
                                  JFrame.EXIT_ON_CLOSE);
      frame.setSize(WIDTH, HEIGHT);
      frame.setVisible(true);
   }

   public static void main(String[] args) {

      javax.swing.SwingUtilities.invokeLater(new
                                              Runnable() {
         public void run() {
            createAndShowGUI();
         } });
   }
}
```

2) (max. 20 Punkte)

Das obige Programm verfügt bereits über einen Button, der jedoch nichts Wesentliches bewirkt. Ergänzen Sie die Applikation um einen Event-Handler, der dafür sorgt, dass die Animation aus Aufgabe 2 gestoppt wird.

Hinweise:
Es ist ein Unterschied, ob der Animations-Thread terminiert (max. 20 Punkte erreichbar) oder ob es nur so aussieht, als wäre die Animation beendet (max. 15 Punkte erreichbar).
Falls es Ihnen nicht gelungen ist, Aufgabe 1 zu lösen, können Sie folgende Variante bearbeiten (dabei sind max. 10 Punkte erreichbar):
Ergänzen Sie die Applikation um einen Event-Handler, der dafür sorgt, dass sich die Position des Kreises bei Knopfdruck verändert.

3) (max. 10 Punkte)

Motivation: Die Klasse java.awt.Color hat einen Konstruktor Color(int r, int, g, int b) für RGB-Werte, die im Bereich 0 bis 255 (einschließlich der Grenzen) liegen müssen.

Schreiben Sie eine Methode, die unter Verwendung von einer oder mehreren Bitoperationen prüft, ob ein übergebener int-Parameter einen Wert im Bereich zwischen 0 und 255 hat. Falls dies nicht der Fall ist, soll eine java.lang.IllegalArgumentException geworfen werden.

4) (max. 20 Punkte)

Gegeben sei die folgende einfache Klasse S.

```
class S {

    String s;

    public S(String s){
        this.s = s;
    }

    public String toString(){
        return s;
    }
}
```

A) Schreiben Sie eine Testklasse, in deren main-Methode die folgenden Anforderungen erfüllt werden.

1. Erzeugen Sie 3 Instanzen von S und speichern Sie sie in einer verzeigerten Liste (java.util.LinkedList<E>).

2. Geben Sie danach alle Einträge der Liste mit System.out.println() aus. Verwenden Sie dazu die mit Java 1.5 eingeführte foreach-Anweisung.

3. Geben Sie alle Einträge nochmals aus. Diesmal unter Verwendung des Iterators, den Sie mit der nachfolgend aufgeführten Methode erhalten können

   public ListIterator<E> listIterator(int index)
   Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list. Obeys the general contract of List.listIterator(int).

B) Stellen Sie dar, wie man vor der Einführung von Java 1.5 den Teil A hätte bearbeiten müssen.

Aufgabe 122

- Eine neue Klasse erstellen, die von Thread erbt:

~~import~~

```
class AnimationThread extends Thread {

    private ~~static~~ ~~int~~ MyPaintPanel panel;        // Referenz auf zu beeinflussenden Panel
    private volatile boolean keepRunning;                // Laufbedingung

    public void run () {

        while (keepRunning) {                            // Abfrage der Laufbedingung

            panel.repaint();                             // Panel neu zeichnen lassen
            panel.setNewCirclePosition();                // neue Position setzen,
                                                         // wird im nächsten Durchlauf genommen

            try {

                sleep (1000/24);                         // 1/24 Sekunde schlafen

            } catch (InterruptedException ex) {          // falls mit Interrupt() unterbrochen

                System.err.println (ex.getMessage());

            } // catch

        } // while

    } // run()

    /* Konstruktor */
    public AnimationThread (MyPaintPanel panel) {

        super ("AnimationThread");                       // Superconstructor mit Threadnamen
        this.panel = panel;                              // Referenz auf JPanel/MyPaintPanel setzen
        keepRunning = true;                              // Laufbedingung instanziieren

    } // con()

    public void stopAnimation () {

        keepRunning = false;                             // Laufbedingung auf false setzen

    }

} // class
```

- Erweitern der Ursprungsklasse:

```
// Animation Application

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;                                    ✓

class MyFrame extends JFrame {

    private JButton button;
    private JPanel panel1;
    private ~~JPanel~~ MyPaintPanel panel2;
    private AnimationThread animThread;

    public MyFrame (String str) {
        :
        button = new JButton ("stopp");

        button.addActionListener ( new ActionListener () {  //actionL. auf button
            public void actionPerformed ( ActionEvent e ) {

                MyFrame.this.animThread.stopAnimation();  //beendet (indirekt)
                                                          //   den Thread
                                                          //über Laufbedingung
            }
        });
        :
        animThread = new AnimationThread ( panel2 );  //instanziieren mit
                                                      //Referenz auf Zeichenpanel
        animThread.start();                           // Thread starten

    } // con()
} // class
```

```
class MyPaintPanel extends JPanel {
    :
    public void setNewCirclePosition () {      // Methode (für Thread),
                                               // um x-Position zu
                                               // verändern (Radbewegung)
        xPos++;
    }
} // class
```

Primer ! 40 P

Aufgabe 3

```
class BitChecks {

    public static void checkValidValue ( int i )  //überprüft Bereich
        throws IllegalArgumentException {
            if (  (i & ~255) != 0 )        // Erläuterung siehe unten
                throw new IllegalArgumentException();...
            return;                        // keine else nötig
                                           // sicherheitshalber korrektes
                                           // Beenden der Methode
    }

}// class
```

Überlegung: Bei 255 sind alle Bits von 2^0 bis 2^7 gesetzt. Interessant ist nun, ob ein größerer oder negativer Wert existiert, hier wäre mindestens eines der restlichen Bits gesetzt. Mit dem Komplement von 255 (alle Bits außer $2^0 - 2^7$ gesetzt) ein bitweises UND, nun habe ich eine int-Zahl, die entweder 0 oder negativ oder > 255 ist. Bei 0 wäre die ursprüngliche Zahl im gültigen Bereich, also wird bei != 0 eine Exception geworfen.

10 P

Aufgabe 4 (A)

```
import java.util.*;                                              ✓
class TestS {

    public static void main ( final String[] args) {

        // Erzeugung 3er Instanzen von S und Speichern in
        // einer LinkedList
                                     new LinkedList<S>();
        LinkedList<S> list = (LinkedList<S>) new Object[3];      f
                             //cast nötig, da generisch. Länge/Größe max. 3
        for (int i = 1; i <= 3; i++) {
            list.add (new S("Objekt Nummer " + i));   // 3 neue, anonyme
                                                  ✓   // S-Objekte hinzufügen
        }
                                                      2P


        // Ausgabe der Einträge in list
        for (S s : list)                                 // list implementiert
                                                         // Iterable-Interface, daher
            System.out.println (s);                  ✓   // direktes durchiterieren
                                                         // möglich
                                                              5P

        // Ausgabe der Einträge in list über den ListIterator
        ListIterator<S> it = list.listIterator (0);   // ges. Iterator holen, soll
                                                      // Objekte vom Typ S
        for ( S s : it.next() )            f          // ausgeben

            System.out.println (s);                           0P

    } // main()

} // class
```

Aufgabe 4 (B)

```
import java.util.*;
class TestS {
    public static void main (final String[] args) {
        // Erzeugung 3er Instanzen und Abspeichern
        LinkedList list = new LinkedList(3); // maximal 3 Elemente
        for (int i = 1; i <= 3; i++) {
            list.add (new S ("Objekt Nummer " + i));
                                    // 3x ein neues anon. Objekt von S
                                    // hinzufügen in die Liste
        }

        // Ausgabe der Einträge in list
        for (ListIterator it = list.listIterator(0); it.hasNext(); ) {
                                    // Iterator mit startindex 0 holen, fragen nach Nachfolger
            Object o = it.next();   // Objekt holen
            if (o instanceof S)     // Fragen, ob es wirklich vom Typ S ist
                System.out.println ( (S) o );  // dann den cast ausgeben
        }
    }//main()
}//class
```

V

5 P