GdP−WS−1516−Teil−III

```
Programmbeispiele zum Teil III der Vorlesung

      Grunglagen der Programmierung


              Prof. K.Bothe

                 WS 15/16
```

Stack.java

```
// *** III.1 Grundkonzepte der Objektorientierung (1):
// *** abstrakte Datentypen, Objekte, Klassen
// *** v.30.11.05

class Stack {

    private char [] stackElements;

    private int top;  // zeigt auf oberstes Element

    public Stack(int n) {
        stackElements = new  char [n];
        top = -1;
    }

    public boolean isempty() {
        return top == -1;
    }

    public void push(char x) {
        top++; stackElements[top] = x;
    }

    public char top() {
        if (isempty()) {
          System.out.println("Stack leer");
          return ' ';
        }
        else
          return stackElements [top];
    }

    public void pop() {
        if (isempty())
          System.out.println("Stack leer");
        else
          top--;
    }
}
```

Umkehrung.java

```java
// *** III.1 Grundkonzepte der Objektorientierung (1):
// *** abstrakte Datentypen, Objekte, Klassen
// *** v.30.11.05

public class Umkehrung {

  public static void main (String argv[]) {

    int n;
    char ch;
    Stack s;

    System.out.print("Groesse des Stacks: ");
    n = Keyboard.readInt();
    s = new Stack(n);

      // n Elemente einlesen und in den Stack speichern
    System.out.println("Gib mindestens " + n + " Zeichen ein:");
    for (int i=0; i < n; i++) {
      ch = Keyboard.readChar();
      s.push(ch);
    }

    System.out.println
         ("Umgekehrte Reihenfolge der ersten " + n + " Zeichen:");
    while (!s.isempty()) {                // solange Stack nicht leer:
        System.out.print(s.top());        // drucke und streiche
        s.pop();                          // oberstes Element
    }
    System.out.println();
  }
}
```

Umkehrung2.java

```java
// *** III.1 Grundkonzepte der Objektorientierung (1):
// *** abstrakte Datentypen, Objekte, Klassen
// *** v.30.11.05

public class Umkehrung2 {
   // Demonstration von 2 Stackobjekten

   // Umkehrung der Liste in zwei Abschnitten:
   // 1. die an ungerader Position
   // 2.  ... gerader Position

  public static void main (String argv[]) {

    int n;
    char ch;
    Stack s1, s2;

    System.out.print("Groesse der Stacks: ");
    n = Keyboard.readInt();
    s1 = new Stack(n);
    s2 = new Stack(n);

      // 2 * n Elemente einlesen und in den Stack speichern
    System.out.println("Gib mindestens " + 2 * n + " Zeichen ein:");
    for (int i=0; i < 2 * n; i++) {
      ch = Keyboard.readChar();
      if (i % 2 == 0)
        s1.push(ch);
      else
        s2.push(ch);
    }

    System.out.println
        ("Umgekehrte Reihenfolge der ersten " + 2 * n + " Zeichen");
    System.out.println("in zwei Abschnitten: ");
    System.out.println("1. an ungerader 2. an gerader Position");

    while (!s2.isempty()) {               // solange Stack nicht leer:
        System.out.print(s2.top());       // drucke und streiche
        s2.pop();                         // oberstes Element
    }
    System.out.println();
    while (!s1.isempty()) {               // solange Stack nicht leer:
        System.out.print(s1.top());       // drucke und streiche
        s1.pop();                         // oberstes Element
    }
    System.out.println();
  }
}
```

KlammerStruktur.java

```
// *** III.2 Objektorientierung: Grundlegende Fallbeispiele
// *** v.30.11.05

class KlammerStruktur {

  public static void main(String[] argv) {

    final int N = 100;
    char [] eingabe = new char [N];        // Eingabeprogramm
    int j = 0;              // gefuellt bis zur der Laenge j-1
    int i = 0;              // Index: durchlaeuft das Eingabeprogramm
    boolean ok = true;      // zu Beginn: kein Fehler
    Stack s = new Stack(20);               // Klammerstack

    System.out.println
      ("Ausdruck mit Klammern eingeben: {}, [], () (Ende: .):");
    do {
      eingabe[j] = Keyboard.readChar(); j++;
    }
    while ( eingabe[j-1] != ’.’); // Eingabe endet mit ’.’

    while ((i < j) && (ok)) {   // solange noch Eingabezeichen vorhanden
                                // und kein Fehler aufgetreten ist
      switch (eingabe[i]) {
        case ’(’:               // oeffnende Klammern: abspeichern
        case ’{’:
        case ’[’: s.push(eingabe[i]); break;
        case ’)’: if (!s.isempty() && s.top() == ’(’)
                       s.pop();
                  else ok = false; break;
        case ’}’: if (!s.isempty() && s.top() == ’{’)
                       s.pop();
                  else ok = false; break;
        case ’]’: if (!s.isempty() && (s.top()) == ’[’)
                       s.pop();
                  else ok = false; break;
        default:  break;       // keine Klammer
      }
      i++;
    }
           // Ende-Test: alle Zeichen der Eingabe erfasst und ...
    if ((i==j) && ok && s.isempty())
         System.out.println("Klammerstruktur ok!");
    else System.out.println("Klammerstruktur falsch!");
  }
}
```

Time.java

```
// *** III.2 Objektorientierung: Grundlegende Fallbeispiele
// *** v.30.11.05

class Time {

  private int hour, minute; // die aktuelle Zeit

  public Time (int h, int m) { hour = h; minute = m; }

  public Time () {hour = 0; minute = 0; }

  public void addMinutes (int m) {
    //erhoeht die aktuelle Zeit um m Minuten

    int totalMinutes = (60*hour + minute + m) % (24*60);
    if (totalMinutes < 0)
      totalMinutes = totalMinutes + 24*60;
    hour = totalMinutes/60; minute = totalMinutes%60;
  }

  public void subtractMinutes (int m) {
    addMinutes(-m);
  }

  public void printTime () {

    // druckt die aktuelle Zeit nach
    // englischen Konventionen: AM, PM, noon, midnight

    if ((hour == 0) && (minute == 0))
      System.out.print("midnight");
    else if ((hour == 12) && (minute == 0))
      System.out.print("noon");
    else {
      if      (hour == 0)  System.out.print(12);
      else if (hour > 12)  System.out.print(hour-12);
      else                 System.out.print(hour);

      if (minute < 10) System.out.print(":0" + minute);
      else             System.out.print(":" + minute);

      if (hour < 12) System.out.print("AM");
      else           System.out.print("PM");
    }
  }

  public void printTimeInMinutes () {

     // druckt aktuelle Zeit mit Entsprechung in Minuten

    printTime ();
    System.out.println("  =" + timeInMinutes() + ". Minute des Tages ");
  }

  private  int timeInMinutes () {            // private: Hilfsfunktion

    // ermittelt die Anzahl von Minuten seit 0:00 Uhr,
    // die der aktuellen Zeit entspricht
    int totalMinutes = (60*hour + minute) % (24*60);
    if (totalMinutes < 0)
      totalMinutes = totalMinutes + 24*60;
    return totalMinutes;
  }

  public boolean before (Time t) {
    return ((hour < t.hour) ||
            ((hour == t.hour) && (minute < t.minute)));
  }

  public boolean after (Time t2) { return t2.before(this); }

  public Time copy () {return new Time(hour,minute); }

}
```

```
// *** III.2 Objektorientierung: Grundlegende Fallbeispiele
// *** v.30.11.05

class Schedule {

  private static void includeNewEntry
                       (Time t, String s, int intervalInMinutes) {
    // druckt eine Zeile: Zeitangabe, Name s der Veranstaltung;
    // erhoeht Zeit t um Laenge (intervalInMinutes) der Veranstaltung

    t.printTime();
    System.out.println("  "+ s);
    t.addMinutes(intervalInMinutes);
  }

  public static void main (String[] args) {
    Time t1 = new Time(8,30);
    Time t2 = new Time();
    Time t3, t4;

   // Druck von Terminplaenen:

    System.out.println("erster Plan:");
    includeNewEntry(t1,"PI1",90);
    includeNewEntry(t1,"Pause",15);
    includeNewEntry(t1,"TI1",90);
    includeNewEntry(t1,"Pause",15);
    t3 = t1;                        // verweisen auf dasselbe Objekt
    t4 = t1.copy();                 // ganzes Objekt kopiert
    includeNewEntry(t1,"Ma1",100);
    includeNewEntry(t1,"Pause",15);
    includeNewEntry(t1,"Ma2",20);
    System.out.println("letzte (aktuelle) Tageszeit in Minuten: ");
    t1.printTimeInMinutes ();
    System.out.println(" ");

    System.out.println("zweiter Plan:");
    includeNewEntry(t2,"Nebenfach",100);
    includeNewEntry(t2,"Proseminar",90);
    System.out.println("letzte (aktuelle) Tageszeit in Minuten: ");
    t2.printTimeInMinutes ();
    System.out.println(" ");

    System.out.println("dritter Plan:");     // t1, t3 -> ein Objekt
    includeNewEntry(t3,"Sport",100);          // t3: aktuelle t1-Zeit
    includeNewEntry(t3,"Freizeit",200);
    System.out.println("letzte (aktuelle) Tageszeit in Minuten: ");
    t3.printTimeInMinutes ();
    System.out.println(" ");

    if (t4.before(t1)) {
      System.out.println("vierter Plan:");                 // t1 <> t4
      includeNewEntry(t4,"zweiters Nebenfach",100); // t4: alte t1-Zeit
      includeNewEntry(t4,"frei",200);
      System.out.println("letzte (aktuelle) Tageszeit in Minuten: ");
      t4.printTimeInMinutes ();
    }
  }
}
```

```
// *** III.3 Grundkonzepte der Objektorientierung (2):
// *** Klassenmethoden, Klassenvariablen
// *** v.30.11.05

class TimeC {

  private int hour, minute; // die aktuelle Zeit
  private final static int noonHour = 12;
  private final static int noonMinute = 0;
  private static boolean englishTime = true;

  public TimeC (int h, int m) { hour = h; minute = m; }

  public TimeC () {hour = 0; minute = 0; }

  public static void switchTimeFormat () {
    englishTime = !englishTime;
  }

  public void addMinutes (int m) {
    //erhoeht die aktuelle Zeit um m Minuten

    int totalMinutes = (60*hour + minute + m) % (24*60);
    if (totalMinutes < 0)
      totalMinutes = totalMinutes + 24*60;
    hour = totalMinutes/60; minute = totalMinutes%60;
  }

  public void subtractMinutes (int m) {
    addMinutes(-m);
  }

  public void printTime () {

    if (englishTime)
      printenglishTime();
    else
      printGermanTime();
  }

  private void printenglishTime () {

    // druckt die aktuelle Zeit nach
    // englischen Konventionen: AM, PM, noon, midnight

    if ((hour == 0) && (minute == 0))
      System.out.print("midnight");
    else if ((hour == noonHour) && (minute == noonMinute))
      System.out.print("noon");
    else {
      if      (hour == 0) System.out.print(12);
      else if (hour > 12) System.out.print(hour-12);
      else                System.out.print(hour);

      if (minute < 10) System.out.print(":0" + minute);
      else             System.out.print(":" + minute);

      if (hour < 12) System.out.print("AM");
      else           System.out.print("PM");
```

```
    }
  }

  private void printGermanTime () {

      System.out.print(hour);

      if (minute < 10) System.out.print(":0" + minute);
      else             System.out.print(":" + minute);

  }

  public void printTimeInMinutes () {

     // druckt aktuelle Zeit mit Entsprechung in Minuten

    printTime ();
    System.out.println(" =" + timeInMinutes() + ". Minute des Tages ");
  }

  private  int timeInMinutes () {            // private: Hilfsfunktion

    // ermittelt die Anzahl von Minuten seit 0:00 Uhr,
    // die der aktuellen Zeit entspricht

    int totalMinutes = (60*hour + minute) % (24*60);
    if (totalMinutes < 0)
      totalMinutes = totalMinutes + 24*60;
    return totalMinutes;
  }

  public boolean before (TimeC t) {
    return ((hour < t.hour) ||
            ((hour == t.hour) && (minute < t.minute)));
  }

  public boolean after (TimeC t2) { return t2.before(this); }

  public TimeC copy () {return new TimeC(hour,minute); }

}
```

ScheduleC.java

```
// *** III.3 Grundkonzepte der Objektorientierung (2):
// *** Klassenmethoden, Klassenvariablen, v.30.11.05

class ScheduleC {

  private static void includeNewEntry
                        (TimeC t, String s, int intervalInMinutes) {
    // druckt eine Zeile: Zeitangabe, Name s der Veranstaltung;
    // erhoeht Zeit t um Laenge (intervalInMinutes) der Veranstaltung
    t.printTime();
    System.out.println("  "+ s);
    t.addMinutes(intervalInMinutes);
  }

  public static void main (String[] args) {
    TimeC t1 = new TimeC(8,30);
    TimeC t2 = new TimeC();
    TimeC t3, t4;

   // Druck von Terminplaenen:
    System.out.println("erster Plan:");
    includeNewEntry(t1,"PI1",90);
    includeNewEntry(t1,"Pause",15);
    includeNewEntry(t1,"TI1",90);
    includeNewEntry(t1,"Pause",15);
    t3 = t1;                        // verweisen auf dasselbe Objekt
    t4 = t1.copy();                 // ganzes Objekt kopiert
    includeNewEntry(t1,"Ma1",100);
    includeNewEntry(t1,"Pause",15);
    includeNewEntry(t1,"Ma2",20);
    System.out.println("letzte (aktuelle) Tageszeit in Minuten: ");
    t1.printTimeInMinutes ();
    System.out.println(" ");

    System.out.println("zweiter Plan:");
    includeNewEntry(t2,"Nebenfach",100);
    includeNewEntry(t2,"Proseminar",90);
    System.out.println("letzte (aktuelle) Tageszeit in Minuten: ");
    t2.printTimeInMinutes ();
    System.out.println(" ");

    TimeC.switchTimeFormat();

    System.out.println("dritter Plan:");     // t1, t3 -> ein Objekt
    includeNewEntry(t3,"Sport",100);          // t3: aktuelle t1-Zeit
    includeNewEntry(t3,"Freizeit",200);
    System.out.println("letzte (aktuelle) Tageszeit in Minuten: ");
    t3.printTimeInMinutes ();
    System.out.println(" ");

    if (t4.before(t1)) {
      System.out.println("vierter Plan:");                  // t1 <> t4
      includeNewEntry(t4,"zweiters Nebenfach",100); // t4: alte t1-Zeit
      includeNewEntry(t4,"frei",200);
      System.out.println("letzte (aktuelle) Tageszeit in Minuten: ");
      t4.printTimeInMinutes ();
    }
  }
}
```

StackForChar.java

```
// *** III.5 Grundkonzepte der Objektorientierung (3):
// *** Vererbung, Polymorphismus
// *** v.30.11.05

class Stack {

    private Object [] stackElements;
    private int top;

    public Stack(int n) {
        stackElements = new  Object [n]; top = -1;
    }

    public boolean isempty() {
        return top == -1;
    }

    public void push(Object x) {
        top++; stackElements[top] = x;
    }

    public Object top() {
        if (isempty()) {
          System.out.println("Stack leer");
          return null;
        }
        else
          return stackElements [top];
    }

    public void pop() {
        if (isempty()) System.out.println("Stack leer");
        else
          top--;
    }
}

class Char {
  char c;
  public Char (char ch) {
    c = ch;
  }
  public char charValue () {
    return c;
  }
}

public class StackForChar {

  public static void main (String argv[]) {

    int n;
    Char ch;
    Stack s;

    System.out.print("Groesse des Stacks: ");
    n = Keyboard.readInt();
    s = new Stack(n);
```

```
      // n Elemente einlesen und in den Stack speichern
    System.out.println("Gib mindestens " + n + " Zeichen ein:");
    for (int i=0; i < n; i++) {
      ch = new Char(Keyboard.readChar());
      s.push(ch);
    }

    System.out.println
        ("Umgekehrte Reihenfolge der ersten " + n + " Zeichen:");
    while (!s.isempty()) {              // solange Stack nicht leer:
        Char c = (Char) s.top();
        System.out.print(c.charValue());
        s.pop();
    }
    System.out.println();
  }
}
```

```
// *** III.5 Grundkonzepte der Objektorientierung (3):
// *** Vererbung, Polymorphismus
// *** v.30.11.05

class Time {
  protected int
    hour, minute;

  public Time (int h, int m) {
    hour = h;
    minute = m;
  }

  public void addMinutes (int m) {
    int totalMinutes = (60*hour + minute + m)%(24*60);
    if (totalMinutes < 0)
      totalMinutes = totalMinutes + 24*60;
    hour = totalMinutes/60;
    minute = totalMinutes%60;
  }

  public void printTime () {
    if ((hour == 0) && (minute == 0))
      System.out.print("midnight");
    else if ((hour == 12) && (minute == 0))
      System.out.print("noon");
    else {
      if (hour == 0)
        System.out.print(12);
      else if (hour > 12)
        System.out.print(hour-12);
      else
        System.out.print(hour);
      if (minute < 10) System.out.print(":0" + minute);
      else             System.out.print(":" + minute);
        if (hour < 12)
          System.out.print("AM");
        else
          System.out.print("PM");
    }
  }
}

class PreciseTime extends Time {
  private int second;

  public PreciseTime (int h, int m, int s) {
    super(h, m);
    second = s;
  }

  public void addSeconds (int s) {
    int advMinutes = s / 60;
    second += s % 60;
    if (second < 0) {
      advMinutes--;
      second += 60;
    }
    else if (second >= 60) {
```

Time2.java

```
      advMinutes++;
      second -= 60;
    }
    addMinutes(advMinutes);
  }

  public void printTime () {
    if ((hour == 0) && (minute == 0))
      System.out.print("midnight");
    else if ((hour == 12) && (minute == 0))
      System.out.print("noon");
    else {
      if (hour == 0)
        System.out.print(12);
      else if (hour > 12)
        System.out.print(hour-12);
      else
        System.out.print(hour);
      if (minute < 10) System.out.print(":0" + minute);
      else             System.out.print(":" + minute);
      if (second < 10) System.out.print(":0" + second);
      else             System.out.print(":" + second);
      if (hour < 12)
        System.out.print("AM");
      else
        System.out.print("PM");
    }
  }
}

class Time2 {
  public static void main(String[] args) {
    PreciseTime lunchtime = new PreciseTime(12, 1, 0);
    lunchtime.addMinutes(1);
    lunchtime.printTime(); System.out.println();
    lunchtime.addSeconds(-61);
    lunchtime.printTime(); System.out.println();
    lunchtime.addSeconds(1);
    lunchtime.printTime(); System.out.println();
  }
}
```

BuildPairs.java

```
// *** III.6 Grundkonzepte der Objektorientierung (4):
// *** Generische Klassen
// *** v.30.11.05

class Pair <T> {
  private T first;
  private T second;

  Pair(T fst, T scd) {
     first = fst;
     second = scd;
  }

  public T getFirst() {
    return first;
  }

  public T getSecond() {
    return second;
  }
}

class BuildPairs {

  public static void main (String[] args) {
    Pair<Integer> pi;
    Pair<String>  ps;
    Integer i, j;

    i = new Integer(99);
    j = new Integer(100);
    pi =
      new Pair<Integer> (i, j);
    ps =
      new Pair<String> ("Hallo", "World");

    System.out.println(ps.getFirst() + " " + ps.getSecond());
    System.out.println(pi.getFirst().intValue()
           + " " + pi.getSecond().intValue());
  }
}
```

BuildPairsBounds.java

```
// *** III.6 Grundkonzepte der Objektorientierung (4):
// *** Generische Klassen
// *** v.30.11.05

class PairNumber <T extends Number> {
  private T first;
  private T second;

  PairNumber(T fst, T scd) {
     first = fst;
     second = scd;
  }

  public T getFirst() {
    return first;
  }

  public T getSecond() {
    return second;
  }

 public double add () {
    return first.doubleValue() + second.doubleValue();
  }
}

class BuildPairsBounds {

  public static void main (String[] args) {
    PairNumber<Integer> pi;
//  PairNumber<String>  ps;  FALSCH
    Integer i, j;

    i = new Integer(99);
    j = new Integer(100);
    pi =
      new PairNumber<Integer> (i, j);

    System.out.println(pi.getFirst().intValue()
           + " " + pi.getSecond().intValue() + " " + pi.add());
  }
}
```

StackGen.java

```
// *** III.6 Grundkonzepte der Objektorientierung (4):
// *** Generische Klassen
// *** v.01.12.05

import java.util.ArrayList;

import utilities.Keyboard;

class Stack <T> {

    private ArrayList<T> stackElements;
    private int top;

    public Stack() {
        stackElements = new  ArrayList<T>();
        top = -1;
    }

    public boolean isempty() {
        return top == -1;
    }

    public void push(T x) {
        top++;
        if (stackElements.size() <= top) {
                stackElements.add(top, x);
        }
        else {
                stackElements.set(top, x);
        }
    }

    public T top() {
        if (isempty()) {
          System.out.println("Stack is empty");
          return null;
        }
        else
          return stackElements.get(top);
    }

    public void pop() {
        if (isempty())
                System.out.println("Stack is empty");
        else
          top--;
    }
}

public class StackGen {

  public static void main (String argv[]) {

    int n;
    Character ch;
    Stack <Character> s;

    System.out.print("Size of the Stack: ");
    n = Keyboard.readInt();
    s = new Stack<Character>();

      // read n elements and store into the stack
    System.out.println("Enter " + n + " characters:");
    for (int i=0; i < n; i++) {
      ch = Character.valueOf(Keyboard.readChar());
      s.push(ch);
    }
    Keyboard.readString();

    System.out.println("Reverse order of the sequence:");
    while (!s.isempty()) {
        System.out.print(s.top());
        s.pop();
    }
    System.out.println();

  }
}
```

```
// *** III.7 Verkettete Strukturen: Listen
// *** v.30.11.05

class IntList {
  private int value;
  private IntList rest;

  public IntList (int v, IntList next) {
    value = v;
    rest = next;
  }

  public int getValue () { return value; }

  public void setValue (int val) { value = val; }

  public IntList getRest () { return rest; }

  public int length () {
    if (rest == null)
      return 1;
    else
      return 1 + rest.length();
  }

  public String toString () {
    String myValue = Integer.toString(value);
    if (rest == null)
      return myValue;
    else
      return myValue + "," + rest.toString();
  }

  public IntList find (int key) {
    if (value == key)
      return this;
    else if (rest == null)
      return null;
    else
      return rest.find(key);
  }

  public IntList nth (int n) {
    if (n == 0)
      return this;
    else if (rest == null)
      return null;
    else
      return rest.nth(n-1);
  }

  public void addToEndM (int val) {
    if (rest != null)
      // a cell in the middle of the list
      rest.addToEndM(val);
    else // the last cell
      rest = new IntList(val, null);
  }
```

IntList.java

```
  public IntList reverseM () {
    return reverseM(null);
  }

  private IntList reverseM (IntList prev) {
    if (rest == null) {
      rest = prev;
      return this;
    }
    else {
      IntList front = rest.reverseM(this);
      rest = prev;
      return front;
    }
  }

  public IntList addInorderM (int n) {
    if (n < value)
      return new IntList (n, this);
    else if (n == value)
      return this;
    else if (rest == null) {
      rest = new IntList(n, null);
      return this;
    }
    else {
      rest = rest.addInorderM(n);
      return this;
    }
  }

  public IntList remove (int n) {
    if (value == n)
      return rest;
    else if (rest == null)
      return this;
    else
      return new IntList(value, rest.remove(n));
  }

}
```

List.java

```
// *** III.7 Verkettete Strukturen: Listen
// *** v.30.11.05

class List {
  public static void main (String[] args) {
    IntList list = new IntList(57, null);
    list = new IntList(1, list);
    list = new IntList(11, list);
    list = new IntList(2, list);
    IntList temp;
    for (temp = list;
             temp != null;
                 temp = temp.getRest())
      System.out.println(temp.getValue() + ",");
    System.out.println();
  }
}
```

```
// *** III.7 Verkettete Strukturen: Listen
// *** v.30.11.05

public class Stack1 {

  // 1. mit verketteten linearen Listen
  // 2. damit: keine Beschraenkung der Groesse
  // 3. lokale Klasse einer Klasse

    private class Zelle {
        Object inhalt; // Inhalt
        Zelle next;    // Verweis
    }

    // Verweis auf oberste Zelle
    private Zelle top;

    public Stack() {
        top = null;
    }

    public boolean isempty() {
        return top == null;
    }

    public void push(Object x) {
        Zelle neueZelle = new Zelle();
        neueZelle.inhalt = x;
        neueZelle.next   = top;
        top           = neueZelle;
    }

    public Object top() {
        if (isempty()) {
           System.out.println("Stack leer");
           return new Character(' ');
        }
        return top.inhalt;
    }

    public void pop() {
        if (isempty())
          System.out.println("Stack leer");
        else
          top = top.next;
    }
}
```

```
// *** III.8 Grundkonzepte der Objektorientierung (5):
// *** Interface
// *** v.30.11.05

interface TimeI {
  public void addMinutes (int m);
  public void subtractMinutes (int m);
  public void printTime ();
  public void printTimeInMinutes () ;
}

class Time implements TimeI{

  private int hour, minute; // die aktuelle Zeit

  public Time (int h, int m) { hour = h; minute = m; }

  public Time () {hour = 0; minute = 0; }

  public void addMinutes (int m) {
    //erhoeht die aktuelle Zeit um m Minuten

    int totalMinutes = (60*hour + minute + m) % (24*60);
    if (totalMinutes < 0)
      totalMinutes = totalMinutes + 24*60;
    hour = totalMinutes/60; minute = totalMinutes%60;
  }

  public void subtractMinutes (int m) {
    addMinutes(-m);
  }

  public void printTime () {

    // druckt die aktuelle Zeit nach
    // englischen Konventionen: AM, PM, noon, midnight

    if ((hour == 0) && (minute == 0))
      System.out.print("midnight");
    else if ((hour == 12) && (minute == 0))
      System.out.print("noon");
    else {
      if      (hour == 0)  System.out.print(12);
      else if (hour > 12)  System.out.print(hour-12);
      else                 System.out.print(hour);

      if (minute < 10) System.out.print(":0" + minute);
      else             System.out.print(":" + minute);

      if (hour < 12) System.out.print("AM");
      else           System.out.print("PM");
    }
  }

  public void printTimeInMinutes () {

     // druckt aktuelle Zeit mit Entsprechung in Minuten

    printTime ();
```

ScheduleInt.java

```
    System.out.println(" =" + timeInMinutes() + ". Minute des Tages ");
  }

  private  int timeInMinutes () {             // private: Hilfsfunktion

    // ermittelt die Anzahl von Minuten seit 0:00 Uhr,
    // die der aktuellen Zeit entspricht

    int totalMinutes = (60*hour + minute) % (24*60);
    if (totalMinutes < 0)
      totalMinutes = totalMinutes + 24*60;
    return totalMinutes;
  }
}

class ScheduleInt {

  private static void includeNewEntry
                         (TimeI t, String s, int intervalInMinutes) {

    // druckt eine Zeile: Zeitangabe, Name s der Veranstaltung;
    // erhoeht Zeit t um Laenge (intervalInMinutes) der Veranstaltung

    t.printTime();
    System.out.println("  "+ s);
    t.addMinutes(intervalInMinutes);
  }


  public static void main (String[] args) {
    TimeI t1 = new Time(8,30);

   // Druck von Terminplaenen:

    System.out.println("erster Plan:");
    includeNewEntry(t1,"PI1",90);
    includeNewEntry(t1,"Pause",15);
    System.out.println("letzte (aktuelle) Tageszeit in Minuten: ");
    t1.printTimeInMinutes ();
    System.out.println(" ");
    }
}
```

ScheduleAbstr.java

```
// *** III.8 Grundkonzepte der Objektorientierung (5):
// *** Interface
// *** v.30.11.05

abstract class TimeI {
  public abstract void addMinutes (int m);
  public abstract void subtractMinutes (int m);
  public abstract void printTime ();
  public abstract void printTimeInMinutes () ;
}

class Time extends TimeI{

  private int hour, minute; // die aktuelle Zeit

  public Time (int h, int m) { hour = h; minute = m; }

  public Time () {hour = 0; minute = 0; }

  public void addMinutes (int m) {
    //erhoeht die aktuelle Zeit um m Minuten

    int totalMinutes = (60*hour + minute + m) % (24*60);
    if (totalMinutes < 0)
      totalMinutes = totalMinutes + 24*60;
    hour = totalMinutes/60; minute = totalMinutes%60;
  }

  public void subtractMinutes (int m) {
    addMinutes(-m);
  }

  public void printTime () {

    // druckt die aktuelle Zeit nach
    // englischen Konventionen: AM, PM, noon, midnight

    if ((hour == 0) && (minute == 0))
      System.out.print("midnight");
    else if ((hour == 12) && (minute == 0))
      System.out.print("noon");
    else {
      if      (hour == 0) System.out.print(12);
      else if (hour > 12) System.out.print(hour-12);
      else                System.out.print(hour);

      if (minute < 10) System.out.print(":0" + minute);
      else             System.out.print(":" + minute);

      if (hour < 12) System.out.print("AM");
      else           System.out.print("PM");
    }
  }

  public void printTimeInMinutes () {

     // druckt aktuelle Zeit mit Entsprechung in Minuten

    printTime ();
```

ScheduleAbstr.java

```
    System.out.println(" =" + timeInMinutes() + ". Minute des Tages ");
  }

  private  int timeInMinutes () {             // private: Hilfsfunktion

    // ermittelt die Anzahl von Minuten seit 0:00 Uhr,
    // die der aktuellen Zeit entspricht

    int totalMinutes = (60*hour + minute) % (24*60);
    if (totalMinutes < 0)
      totalMinutes = totalMinutes + 24*60;
    return totalMinutes;
  }
}

class ScheduleAbstr {

  private static void includeNewEntry
                       (TimeI t, String s, int intervalInMinutes) {

    // druckt eine Zeile: Zeitangabe, Name s der Veranstaltung;
    // erhoeht Zeit t um Laenge (intervalInMinutes) der Veranstaltung

    t.printTime();
    System.out.println("  "+ s);
    t.addMinutes(intervalInMinutes);
  }


  public static void main (String[] args) {
    TimeI t1 = new Time(8,30);

   // Druck von Terminplaenen:

    System.out.println("erster Plan:");
    includeNewEntry(t1,"PI1",90);
    includeNewEntry(t1,"Pause",15);
    System.out.println("letzte (aktuelle) Tageszeit in Minuten: ");
    t1.printTimeInMinutes ();
    System.out.println(" ");
    }
}
```

UmkehrungNU.java

```
// *** III.7 Objektorientierte Programmierung (4):
// *** Interface

interface Stack {

  public boolean isempty ();
  public void push(char x);
  public char top();
  public void pop();
}

class StackN implements Stack {

    private char [] stackElements;
    private int top;  // zeigt auf oberstes Element

    public StackN(int n) {
        stackElements = new  char [n]; top = -1;
    }

    public boolean isempty() {
        return top == -1;
    }

    public void push(char x) {
        top++; stackElements[top] = x;
    }

    public char top() {
        if (isempty()) {
          System.out.println("Stack leer");
          return ' ';
        }
        else
          return stackElements [top];
    }

    public void pop() {
        if (isempty()) System.out.println("Stack leer");
        else
          top--;
    }
}

class StackU implements Stack {

    private class Zelle {
        char inhalt; // Inhalt
        Zelle next;    // Verweis
    }
    // Verweis auf oberste Zelle
    private Zelle top;

    public StackU() {
        top = null;
    }

    public boolean isempty() {
        return top == null;
    }

    public void push(char x) {
        Zelle neueZelle = new Zelle();
        neueZelle.inhalt = x;
        neueZelle.next   = top;
        top          = neueZelle;
    }

    public char top() {
        if (isempty()) {
           System.out.println("Stack leer");
           return ' ';
        }
        return top.inhalt;
    }

    public void pop() {
        if (isempty())
          System.out.println("Stack leer");
        else
          top = top.next;
    }
}

public class UmkehrungNU {

  public static void main (String argv[]) {

    int n = 10;
    char ch;
    char jn;
    Stack s;

    System.out.print("Stack begrenzt: (j/n)");
    jn = Keyboard.readChar();
    if (jn == 'j') {
      Keyboard.readChar(); // skip newline
      System.out.println("Groesse des Stacks: ");
      n = Keyboard.readInt();
      s = new StackN(n);
    }
    else
      s = new StackU();
      // n Elemente einlesen und in den Stack speichern
    System.out.println("Gib mindestens " + n + " Zeichen ein:");
    for (int i=0; i < n; i++) {
      ch = Keyboard.readChar();
      s.push(ch);
    }

    System.out.println ("Umgekehrte Reihenfolge der ersten " + n + " Zeichen:");
    while (!s.isempty()) {              // solange Stack nicht leer:
        System.out.print(s.top());      // drucke und streiche
        s.pop();                        // oberstes Element
    }
    System.out.println();
  }
}
```

```
// *** III.8 Grundkonzepte der Objektorientierung (5):
// *** Interface, v.30.11.05

import java.io.*;

interface KeyboardI {
  public int readInt ();
  public char readChar ();
  public double readDouble ();
  public String readString ();
  public boolean eof ();
}

class Keyboard implements KeyboardI{

// Author: M. Dennis Mickunas, June 9, 1997
// Primitive Keyboard input of integers, reals, strings, and characters.

static boolean iseof = false;
static char c;
static int i;
static double d;
static String s;

/* WARNING:  THE BUFFER VALUE IS SET TO 1 HERE TO OVERCOME
** A KNOWN BUG IN WIN95 (WITH JDK 1.1.3 ONWARDS)
*/
static BufferedReader input
       = new BufferedReader (
              new InputStreamReader(System.in),1);

  public int readInt () {
    if (iseof) return 0;
    System.out.flush();
    try {
      s = input.readLine();
    }
    catch (IOException e) {
      System.exit(-1);
    }
    if (s==null) {
      iseof=true;
      return 0;
    }
    i = new Integer(s.trim()).intValue();
    return i;
  }

  public char readChar () {
    if (iseof) return (char)0;
    System.out.flush();
    try {
      i = input.read();
    }
    catch (IOException e) {
      System.exit(-1);
    }
    if (i == -1) {
      iseof=true;
```

KeyboardIApp.java

```
      return (char)0;
    }
    return (char)i;
  }

  public double readDouble () {
    if (iseof) return 0.0;
    System.out.flush();
    try {
      s = input.readLine();
    }
    catch (IOException e) {
      System.exit(-1);
    }
    if (s==null) {
      iseof=true;
      return 0.0;
    }
    d = new Double(s.trim()).doubleValue();
    return d;
  }

  public String readString () {
    if (iseof) return null;
    System.out.flush();
    try {
      s=input.readLine();
    }
    catch (IOException e) {
      System.exit(-1);
    }
    if (s==null) {
      iseof=true;
      return null;
    }
    return s;
  }

  public boolean eof () {
    return iseof;
  }

}

public class KeyboardIApp {
  public static void main (String argv[]) {

    int n = 10;
    char ch;
    char jn;
    Keyboard kb = new Keyboard();

    System.out.print("Stack begrenzt: (j/n)");
    jn = kb.readChar();
    kb.readChar(); // skip newline
    System.out.println("Groesse des Stacks: ");
    n = kb.readInt();
   }
}
```

Druck.java

```
// *** III.8 Grundkonzepte der Objektorientierung (5):
// *** Interface
// *** v.30.11.05

class Druck {

static void druckeKurve (double x0, double x1,
   double delta,  Function f) {
// drucke die Funktion f an den
// Stellen x0, x0+delta, ... x1
// hier: nur 2 Werte berechnet
System.out.print(f.apply(x0) + " ");
System.out.println(f.apply(x1));
}

public static void main
  (String [] args) {
 SineFunction sinus = new SineFunction();
 CosFunction cosinus = new CosFunction();
 druckeKurve(0,Math.PI/2,0.1,sinus);
 druckeKurve(0,Math.PI/2,0.1,cosinus);
 }
}

interface Function {
  double apply (double x);
}

class SineFunction implements Function {
  public double apply (double x) {
   return Math.sin(x);
  }
}

class CosFunction implements Function {
  public double apply (double x) {
   return Math.cos(x);
  }
}
```

Ausnahme.java

```
// *** III.9 Ausnahmebehandlung
// *** v.30.11.05

class Ausnahme {

    static int makeIntFromString (String s) {
          return Integer.parseInt(s);
    }

    public static void main(String[] args) {
        int i = makeIntFromString(args[0]);
        System.out.println("i=" + i);
    }
}
```

TryCatch.java

```
// *** III.9 Ausnahmebehandlung
// *** v.30.11.05

class TryCatch {
    public static void main(String[] args) {
        try {
            int i = Integer.parseInt(args[0]);
            System.out.println("i=" + i);
        } catch (NumberFormatException e) {
            System.out.println("Als Kommandozeilen−Argument wird ein int−Wert benötigt");
        }
    }
}
```

TryCatchAll.java

```
// *** III.9 Ausnahmebehandlung
// *** v.30.11.05

class TryCatchAll {
  public static void main(String[] args) {
    try {
      int i = Integer.parseInt(args[2]);
      System.out.println("i=" + i);
     } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Aufruf mit einem Parameter");
        System.out.println("Falscher Index: "+ e.getMessage());
        e.printStackTrace();
     } catch (NumberFormatException e) {
        System.out.println("als Argument: int−Wert");
     } catch (Throwable e) {
        e.printStackTrace();
     }
     System.out.println("Programm ordentlich beendet");
  }
}
```

Finally.java

```
// *** III.9 Ausnahmebehandlung
// *** v.30.11.05

class Finally {
  public static void main(String[] args) {
    int i = 1000, j = 0;
    try {
      i /= j;
    } catch (ArithmeticException e) {
        System.out.println(e);
    } finally {
        System.out.println(i + "/0 undef.");
    }
  }
}
```

TryInTry.java

```java
// *** III.9 Ausnahmebehandlung
// *** v.30.11.05

class TryInTry {
    public static void main(String[] args) {
        try {
            try {
                int x = Integer.parseInt(args[0]);
            } catch (NumberFormatException e) {
                System.out.println("Innen: " + e);
            }
        } catch (Throwable e) {
            System.out.println("Aussen: " + e);
        }
    }
}
```

KeyboardTry.java

```java
// *** III.9 Ausnahmebehandlung
// *** v.30.11.05

class KeyboardTry {
 public static int intEinlesen() {
   int zahl = 0;  boolean ok = false;
   System.out.print("Zahl eingeben: ");
   while (! ok) {
     try {
       ok = true;
       zahl = Keyboard.readInt();
     } catch (NumberFormatException e) {
        System.out.print("Keine Zahl!! Noch einmal:");
        ok = false;
     }
   }
   return zahl;
}

 public static void main(String args[]) {
   int i = intEinlesen();
   System.out.println("i = " + i);
 }
}
```

Oeffnungszeit.java

```
// *** III.9 Ausnahmebehandlung
// *** v.30.11.05

import java.util.*;

class Wochenende extends Exception {
        Wochenende(String text) {
            super(text);
        }
}

class Oeffnungszeit {

    static void heuteOffen () throws Wochenende {
        int tag = Calendar.getInstance().get(Calendar.DAY_OF_WEEK);
        if (tag == Calendar.SUNDAY)             // 1
            throw new Wochenende("Sonntag");
        if (tag == Calendar.SATURDAY)           // 7
            throw new Wochenende("Samstag");
        System.out.println("Heute geoeffnet.");
    }

    public static void main(String[] args) {
        try {
           heuteOffen();
        } catch (Wochenende ex) {
            System.out.println(ex.getMessage() + "s geschlossen.");
        }
    }
}
```

Maze.java

```
// *** III.12 Vom Entwurf zur Implementation
// *** v.30.11.05

import java.awt.*;

class Maze {
  // Representation of a maze object

  final int NORTH=0, EAST=1, SOUTH=2, WEST=3;

  // record the walls to the east
  private boolean[][] eWall =
      {{true ,false,false,false,true },
       {false,true ,true ,false,true },
       {true ,true ,false,true ,true }};
  // record the walls to the south
  private boolean[][] sWall =
      {{true ,true ,true ,false},
       {false,false,false,true },
       {false,false,false,false},
       {true ,true ,true ,true }};
  private int height = 3, width = 4;
  private Point size = new Point(width, height);

  // Where is the starting location?
  public Point getStartLocation() {return new Point(0,2);}
  // In which direction do you face to enter?
  public int getStartDirection() {return EAST;}

  public Point getSize() {return size;}

  // Is a given position outside the maze?
  public boolean outside (Point pos) {
    return ((pos.x < 1)
         || (pos.x > width)
         || (pos.y < 1)
         || (pos.y > height)
       );
  }

  // Is there a wall to the 'dir' direction
  // of location (row,col)?
  public boolean checkWall(int dir, int col, int row) {
    switch (dir) {
      case NORTH: return sWall[row-1][col-1];
      case SOUTH: return sWall[row][col-1];
      case EAST:  return eWall[row-1][col];
      default:    return eWall[row-1][col-1];
    }
  }

  // Alternative version of checkWall
  public boolean checkWall(int dir, Point location) {
    return checkWall(dir, location.x, location.y);
  }
}
```

```
// *** III.12 Vom Entwurf zur Implementation
// *** v.30.11.05

import java.awt.*;


public class MazeTest {

  final static int NORTH=0, EAST=1, SOUTH=2, WEST=3;

  public static void main (String[] args) {
     // create a maze
    Maze theMaze = new Maze();

    Point mazeSize = theMaze.getSize();

     // erwartete Resultate :  "+" tatsaechliche Resultate
    System.out.println("start location is (0,2):"
                       +   theMaze.getStartLocation());
    System.out.println("start direction is EAST = 1: "
                       + theMaze.getStartDirection());
    System.out.println("outside true: "
                       +   theMaze.outside (new Point(0,2)));
    System.out.println("not outside -> false:"
                       + theMaze.outside (new Point(4,3)));

    System.out.println("not outside -> false: "
                       + theMaze.outside(new Point(2,2)));
    System.out.println("wall -> true: "
                       + theMaze.checkWall(NORTH, 1, 1));
    System.out.println("no wall -> false: "
                       + theMaze.checkWall(SOUTH, 1, 1));
    System.out.println("wall -> true: "
                       + theMaze.checkWall(EAST,4,1));
 }
}
```

```
// *** III.12 Vom Entwurf zur Implementation
// *** v.30.11.05

import java.awt.*;

class Mouse { // A mouse that can navigate a maze

  final int NORTH=0, EAST=1, SOUTH=2, WEST=3;

  private Maze theMaze;
  public boolean started = false; // true once the maze is entered.
  private Point location; // The location of this Mouse
  private int direction;  // The direction this Mouse is facing

  public Point getLocation () {return location;}

  public Mouse(Maze m) {
    // Where do I start?
    location = m.getStartLocation();
    printoutMove("starting Point " +
          "[" + location.x + "," + location.y + "]");
    // In what direction do I face initially?
    direction = m.getStartDirection();
    theMaze = m;  // my maze!
  }

  public static void printoutMove(String move) {
   System.out.println(" " + move + " ");
  }

  public boolean outsideMaze () { // Am I outside the maze?
    return theMaze.outside(location);
  }

  public boolean facingWall () {
    return theMaze.checkWall(direction, location);
  }

  public void stepForward() {
    switch (direction) {
      case NORTH: location.y--; break;
      case EAST:  location.x++; break;
      case SOUTH: location.y++; break;
      case WEST:  location.x--; break;
    }
      printoutMove("step forward to point" +
          "[" + location.x + "," + location.y + "]");
  }

  public void turnLeft() {
    printoutMove("turn to the left");
    direction = (direction+3) % 4;
  }

  public void turnRight() {
    printoutMove("turn to the right");
    direction = (direction+1) % 4;
  }
}
```

```
// *** III.12 Vom Entwurf zur Implementation
// *** v.30.11.05

import java.awt.*;
public class MouseMaze {
  // a mouse finds a path through the maze

  final static int NORTH=0, EAST=1, SOUTH=2, WEST=3;

  public static void main (String[] args) {

    // create a maze and a mouse
    Maze theMaze = new Maze();
    Mouse littleMouse = new Mouse(theMaze);

    printMaze(theMaze);

    // move the mouse step by step
    do {
      makeStep(littleMouse);
    }
    while (!littleMouse.outsideMaze());
    }


  private static void makeStep(Mouse m) {
    if (m.started) {
      if (!m.outsideMaze()) {
        m.turnRight();
        while (m.facingWall()) {
          m.turnLeft();
        }
        m.stepForward();
      }
    } else {
      m.stepForward();
      m.started=true;
    }
  }

  private static void printMaze(Maze theMaze) {

    final int WIDTH = 90, HEIGHT = 60, MAGNIFICATION = 1;
    Easel e = new Easel();
    SoftFrame scr = new SoftFrame(WIDTH, HEIGHT);

    // for printing: determine the size of a wall segment
    Point mazeSize = theMaze.getSize();
    int cellW = WIDTH/(mazeSize.x+2),
        cellH = HEIGHT/(mazeSize.y+2);

    // print / draw the maze
    for (int row = 1; row <= mazeSize.y + 1; row++)
      for (int col = 1; col <= mazeSize.x; col++)
        if (theMaze.checkWall(NORTH, new Point(col,row)))
          scr.drawLine1( new Point(col*cellW, row*cellH),
                      new Point((col+1)*cellW, row*cellH));
    for (int row = 1; row <= mazeSize.y; row++)
      for (int col = 1; col <= mazeSize.x+1; col++)
```

```
        if (theMaze.checkWall(WEST, new Point(col,row)))
          scr.drawLine1( new Point(col*cellW, row*cellH),
                      new Point(col*cellW, (row+1)*cellH));

    scr.displaySoftFrame(e, MAGNIFICATION);
  }
}
```

Easel.java

```
// *** III.12 Vom Entwurf zur Implementation
// *** v.30.11.05

import java.awt.*;

public class Easel {

  public void paintEasel (Color[][] frameBuffer, int mag) {

    for (int row = 0;
             row < frameBuffer.length;
             row++) {
      for (int col = 0;
               col < frameBuffer[row].length;
               col++) {
        if (frameBuffer[row][col]==Color.black)
          System.out.print("X");
        else
          System.out.print(" ");
      }
      System.out.println();
    }
  }
}
```

SoftFrame.java

```
// *** III.12 Vom Entwurf zur Implementation
// *** v.30.11.05

import java.awt.*;

class SoftFrame {

  private
    Color[][] frameBuffer;
    int width, height;

  public SoftFrame (int w, int h) {
    width = w;
    height = h;
    frameBuffer = new Color[height][width];
  }

  private void setPixel (int col, int row, Color c) {
      if ((0 <= col && col < width)
          && (0 <= row && row < height))
        frameBuffer[row][col] = c;
    }

  public void clearSoftFrame () {
    for (int col = 0; col < width; col++)
      for (int row = 0; row < height; row++)
        frameBuffer[row][col] = Color.white;
  }

  public void displaySoftFrame (Easel e, int mag) {
    e.paintEasel(frameBuffer, mag);
  }

  private void swap (Point p1, Point p2) {
    int x = p1.x,    y = p1.y;
     p1.x = p2.x; p1.y = p2.y;
     p2.x = x;    p2.y = y;
  }

  public void drawHorizontalLine (int row) {
    for (int col = 0; col < width; col++)
      setPixel(col, row, Color.black);
  }

  public void drawVerticalLine (int col) {
    for (int row = 0; row < height; row++)
      setPixel(col, row, Color.black);
  }

  public void drawLongLine (double m, double b) {
    for (int col = 0; col < width; col++) {
      int row = (int)Math.round(m*col + b);
      setPixel(col, row, Color.black);
    }
  }

  public void drawLine1 (Point p1, Point p2) {
    int row, col;
    if (p1.x == p2.x) { // vertical line
      if (p2.y < p1.y)
        swap(p1, p2); // force p1.y <= p2.y
      for (row = p1.y; row <= p2.y; row++)
        setPixel(p1.x, row, Color.black);
    }
    else {
      double m = (p2.y-p1.y)/(double)(p2.x-p1.x),
             b = p1.y-m*p1.x;
      if (Math.abs(m) < 1.0) {
        if (p2.x < p1.x) // force p1 to left of p2
          swap(p1, p2);
        for (col = p1.x; col <= p2.x; col++) {
          row = (int)Math.round(m*col + b);
          setPixel(col, row, Color.black);
        }
      }
      else { // Math.abs(m) >= 1.0
        if (p2.y < p1.y) // force p1 above p2
          swap(p1, p2);
        for (row = p1.y; row <= p2.y; row++) {
          col = (int)Math.round ((row-b)/m);
          setPixel(col, row, Color.black);
        }
      }
    }
  }

  public void drawLine (Point p1, Point p2) {
  // This version of drawLine works only if
  // p1.x<p2.x, p1.y<p2.y, and the slope of
  // the line is positive and <= 1
    int dx = p2.x-p1.x;
    int dy = p2.y-p1.y;
    int p = 2*dy - dx;
    int x = p1.x,
        y = p1.y;
    setPixel(x, y, Color.black);
    for (x = p1.x+1; x <= p2.x; x++) {
      if (p > 0) y++;
      setPixel(x, y, Color.black);
      if (p < 0)
        p = p + 2*dy;
      else
        p = p + 2*(dy - dx);
    }
  }

  public void drawCircle1 (Point p0, int r) {
    int r2 = r*r;
    for (int x = -r; x <= r; x++) {
      int y = (int)Math.round(Math.sqrt(r2 - x*x));
      setPixel(p0.x+x, p0.y+y, Color.black);
      setPixel(p0.x+x, p0.y-y, Color.black);
    }
  }

  public void drawCircle (Point p0, int r) {
    int r2 = r*r;
```

```
    int y = 0;
    int x = r;
    do {
      setPixel(p0.x+x, p0.y+y, Color.black);
      setPixel(p0.x+y, p0.y+x, Color.black);
      setPixel(p0.x-y, p0.y+x, Color.black);
      setPixel(p0.x-x, p0.y+y, Color.black);
      setPixel(p0.x-x, p0.y-y, Color.black);
      setPixel(p0.x-y, p0.y-x, Color.black);
      setPixel(p0.x+y, p0.y-x, Color.black);
      setPixel(p0.x+x, p0.y-y, Color.black);
      y++;
      x = (int)Math.round(Math.sqrt(r2 - y*y));
    }
    while (y <= x);
  }
}
```

```
// *** III.13 Baeume: effektives Suchen und Sortieren, v.30.11.05

public class Baum {

  String inhalt;
  Baum links, rechts;

  public final static Baum LEER = new Baum();

  public Baum () {
    inhalt = null; links  = null; rechts = null;
  }

  public Baum (String x) {
    this(x, LEER, LEER); }

  public Baum (String s, Baum l, Baum r) {
    inhalt = s; links  = l; rechts = r; }

  public boolean isEmpty () {
    return (inhalt == null);
  }

  public Baum left () {
    if (isEmpty()) System.out.println("kein linker Baum");
    return links;
  }

  public Baum right () {
    if (isEmpty())  System.out.println("kein rechter Baum");
    return rechts;
  }

  public String value () {
    if (isEmpty())  System.out.println("kein Wert");
    return inhalt;
  }

  public void insertSorted(String s) {

     if (isEmpty()) {
         inhalt = s; links  = new Baum(); rechts = new Baum();
     }
     else if (s.compareTo(inhalt) == 0)  ;
     else if (s.compareTo(inhalt) < 0)
         links.insertSorted(s);
     else
         rechts.insertSorted(s);
  }

  public Baum search (String s) {
     if (isEmpty()) return null;
     else if (s.compareTo(inhalt) == 0)
        return this;
     else if (s.compareTo(inhalt) < 0)
        return links.search(s);
     else
        return rechts.search(s);
  }
```

```
  public int lengthTree () {
    if (isEmpty()) return 0;
    else
       return 1 + links.lengthTree() + rechts.lengthTree();
  }
}
```

Traverse.java

```
// *** III.13 Baeume: effektives Suchen und Sortieren
// *** v.30.11.05

public class Traverse {

  public static void inorder(Baum b) {
    if (!b.isEmpty()) {
        inorder (b.left());
        System.out.print(b.value() + " ");
        inorder (b.right());
    }
  }

  public static void preorder(Baum b) {
    if (!b.isEmpty()) {
        System.out.print(b.value() + " ");
        preorder(b.left());
        preorder(b.right());
    }
  }

  public static void postorder(Baum b) {
    if (!b.isEmpty()) {
        postorder(b.left());
        postorder(b.right());
        System.out.print (b.value() + " ");
    }
  }
}
```

TraverseTest.java

```
// *** III.13 Baeume: effektives Suchen und Sortieren
// *** v.30.11.05

public class TraverseTest {

    public static void main(String[] argv) {

        Baum t = new Baum();

        t.insertSorted("Faust");
        t.insertSorted("Licht");
        t.insertSorted("Hof");
        t.insertSorted("Ende");
        t.insertSorted("Hofgarten");
        t.insertSorted("Tisch");
        t.insertSorted("Erde");
        t.insertSorted("Baum");

        System.out.print("Inorder:     ");
        Traverse.inorder(t);
        System.out.println();

        System.out.print("Preorder:    ");
        Traverse.preorder(t);
        System.out.println();

        System.out.print("Postorder:   ");
        Traverse.postorder(t);
        System.out.println();

        System.out.println("Laenge= " + t.lengthTree());
    }
}
```

TempApplet.java

```
// *** III.14 Applets
// *** v.30.11.05

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class TempApplet extends Applet
                        implements ActionListener {
  // Convert from Fahrenheit to Centigrade

  TextField tFahr;
  Label lCent;

  public void init() {

  // Create the TextField and the Label
    tFahr = new TextField(10);
    lCent = new Label(
                 "I’ll tell you what that is in degrees C");

  // Lay out the three Components
    add(new Label("Please type the temperature (deg F): "));
    add(tFahr);
    add(lCent);

  // Register the Component Listener
    tFahr.addActionListener(this);
  }

  // Respond to Action Event: typing in the tFahr TextField
  public void actionPerformed (ActionEvent e) {
    double fahr=0.0,
           cent=0.0;
    fahr = Integer.parseInt(tFahr.getText());
    cent = 5.0 * (fahr - 32) / 9.0;
    lCent.setText(fahr + " deg F is " + cent + " deg C");
  }
}
```

EyesApplet.httml

```
<!-- // *** III.15 Ereignisse (Events): Eyes Applet
     // *** (ein Beispiel zu Applets, Events, Graphics)
     // *** v.30.11.05  -->
<html>
<head>
   <title>Eyes!</title>
</head>
<!-- Browser-Hintergrundfarbe -->
<body bgcolor="white">
<applet CODE="EyesApplet.class" WIDTH=500 HEIGHT=400></applet>
</body>
</html>
```

EyesApplet.java

```
// *** III.15 Ereignisse (Events): Eyes Applet
// *** (ein Beispiel zu Applets, Events, Graphics)
// *** v.30.11.05

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class EyesApplet extends Applet
                  implements MouseMotionListener {

  /**
         *
         */
        private static final long serialVersionUID = 7969890806018360380L;
Point cursor;
  Eyes e1, e2;

  public void init () {
    // Register the Listener.
    addMouseMotionListener(this);
    setSize(500,400);
    setBackground(Color.LIGHT_GRAY);
    e1 = new Eyes(new Point (63,30)); // center of one eye
    e2 = new Eyes(new Point (437,30));// center of the other
    cursor = new Point(250, 2000);    // initial cursor
  }

  public void paint(Graphics g) {
    e1.stare(g, cursor);
    e2.stare(g, cursor);
  }

  public void mouseMoved (MouseEvent e) {
    cursor = e.getPoint();
    repaint();
    }

   // not necessary:
  public void mouseDragged (MouseEvent e) {}
}
```

Eyes.java

```
// *** III.15 Ereignisse (Events): Eyes Applet
// *** (ein Beispiel zu Applets, Events, Graphics)
// *** v.30.11.05

import java.awt.*;

public class Eyes {

  private Point left, right,
              leftPupil, rightPupil;
  private final int
    EYE_RADIUS = 30,
    PUPIL_RADIUS = 10;

  public Eyes(Point c) {
    left = new Point(c.x-EYE_RADIUS-3, c.y);
    right = new Point(c.x+EYE_RADIUS+3, c.y);
  }

  private void fillCircle (Graphics g,
                        Point center, int radius) {
  // Utility method 'fillOval': an abbreviated
  // way to draw a filled circle.
    g.fillOval(center.x-radius, center.y-radius,
               2*radius, 2*radius);
  }

  public void stare (Graphics g, Point cursor) {

    // Draw the white eyes
    g.setColor(Color.WHITE);
    fillCircle(g, left, EYE_RADIUS);
    fillCircle(g, right, EYE_RADIUS);

    // Draw the pupils
    g.setColor(Color.black);
    leftPupil = compute (cursor, left);
    fillCircle(g, leftPupil, PUPIL_RADIUS);
    rightPupil = compute (cursor, right);
    fillCircle(g, rightPupil, PUPIL_RADIUS);
  }

  private Point compute (Point cursor, Point eye) {
  // Compute the location of the pupil, given the
  // locations of the eye and the cursor.
    double d = Math.sqrt((cursor.x-eye.x)*(cursor.x-eye.x)
                       + (cursor.y-eye.y)*(cursor.y-eye.y));
    int r = EYE_RADIUS - PUPIL_RADIUS;
    return new Point (eye.x + (int)((cursor.x-eye.x)*r/d),
                      eye.y + (int)((cursor.y-eye.y)*r/d));
  }
}
```

ThreadBasicTest.java

```
// *** III.16 Parallelitaet: Threads
// *** v.30.11.05

class ThreadA1 extends Thread   {

        public void run()   {
                for (int i = 1; i < ThreadBasicTest.LIMIT; i++)   {
                        System.out.println("A: " + i);
                }
                System.out.println("A done");
        }
}

class ThreadB1 extends Thread   {

        public void run()   {
                for (int i = -1; i > -ThreadBasicTest.LIMIT; i--)   {
                        System.out.println("\t\tB: " + i);
                }
                System.out.println("\t\tB done");
        }
}

public class ThreadBasicTest   {
        static final int LIMIT = 21;
        public static Thread ta;
        public static Thread tb;

        public static void main(String[] args)   {
                ta = new ThreadA1();
                tb = new ThreadB1();
                ta.start();
                tb.start();
                System.out.println("  done...");
        }
}
```

ThreadSleep.java

```
// *** III.16 Parallelitaet: Threads
// *** v.30.11.05

class ThreadA2 extends Thread   {

        public void run()   {
                for (int i = 1; i < ThreadSleep.LIMIT; i++)   {
                        try   {
                                sleep(60);
                        } catch(InterruptedException e) {}
                        System.out.println("A: " + i);
                }
                System.out.println("A done");
        }
}

class ThreadB2 extends Thread   {

        public void run()   {
                for (int i = -1; i > -ThreadSleep.LIMIT; i--)   {
                        try {
                                sleep(40);
                        } catch (InterruptedException e) {}
                        System.out.println("\t\tB: " + i);
                }
                System.out.println("\t\tB done");
        }
}

public class ThreadSleep   {
        static final int LIMIT = 21;
        public static Thread ta;
        public static Thread tb;

        public static void main(String[] args)   {
                ta = new ThreadA2();
                tb = new ThreadB2();
                ta.start();
                tb.start();
                System.out.println("  done...");
        }
}
```

ThreadJoin.java

```
// *** III.16 Parallelitaet: Threads
// *** v.30.11.05

class ThreadA3 extends Thread   {

        public void run()   {
                for (int i = 1; i < ThreadJoin.LIMIT; i++)   {
                        System.out.println("A: " + i);
                }
                System.out.println("A done");
        }
}

class ThreadB3 extends Thread   {

        public void run()   {
                for (int i = -1; i > -ThreadJoin.LIMIT/2; i--)   {
                        System.out.println("\t\tB: " + i);
                }
                try {
                        ThreadJoin.ta.join();
                }catch (InterruptedException e) {}
                System.out.println("\t\tB done");
        }
}

public class ThreadJoin   {
        static final int LIMIT = 21;
        public static Thread ta;
        public static Thread tb;

        public static void main(String[] args)   {
                ta = new ThreadA3();
                tb = new ThreadB3();
                ta.start();
                tb.start();
                System.out.println("  done...");
        }
}
```

ThreadPriority.java

```
// *** III.16 Parallelitaet: Threads
// *** v.30.11.05

class ThreadA4 extends Thread   {

        public void run()   {
                for (int i = 1; i < ThreadPriority.LIMIT; i++)   {
                        System.out.println("A: " + i);
                }
                System.out.println("A done");
        }
}

class ThreadB4 extends Thread   {

        public void run()   {
                for (int i = -1; i > -ThreadPriority.LIMIT; i--)   {
                        System.out.println("\t\tB: " + i);
                        if (i== -1)  {
                                ThreadPriority.ta.setPriority(this.getPriority()
 + 1);
                                System.out.println("Decreased ");
                        }
                }
                System.out.println("\t\tB done");
        }
}

public class ThreadPriority   {
        static final int LIMIT = 21;
        public static Thread ta;
        public static Thread tb;

        public static void main(String[] args)   {
                ta = new ThreadA4();
                tb = new ThreadB4();
                tb.start();
                ta.start();
                System.out.println("  done...");
        }
}
```

SpotTest.httml

```
<!-- // *** III.16 Parallelitaet: Threads
     // *** v.30.11.05  -->
<!DOCTYPE HTML><HTML><HEAD></HEAD>
<BODY>
<APPLET CODE="SpotTest.class" CODEBASE="." WIDTH=400 HEIGHT=300></APPLET>
</BODY>
</HTML>
```

SpotTest.java

```
// *** III.16 Parallelitaet: Threads
// *** v.30.11.05

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class SpotTest extends Applet {

  /* SpotTest                J M Bishop Aug 2000
   * ========
   *
   * Draws spots of different colours
   *
   * Illustrates simple threads
   */

    int mx, my;
    int radius = 10;
    int boardSize = 200;
    int change;

    public void init() {
      boardSize = getSize().width - 1;
      change = boardSize-radius;

      // creates and starts three threads
      new Spots(Color.red).start();
      new Spots(Color.blue).start();
      new Spots(Color.green).start();
    }


  class Spots extends Thread {

    Color colour;

    // the constructor records the thread's colour
        Spots(Color c) {
          colour = c;
        }

    // a very simple run method
    public void run () {
      while (true) {
        draw();
        try {
          sleep (500); // millisecs
        }
        catch (InterruptedException e) {
        }
      }
    }

   public void draw() {
     Graphics g = getGraphics();
     g.setColor(colour);
     // calculate a new place for a spot
     // and draw it.
     mx = (int)(Math.random()*1000) % change;
     my = (int)(Math.random()*1000) % change;
     g.fillOval(mx, my, radius, radius);
   }
  }
}
```