PROJEKT

LAGE FÜR AUTOTEILE

TABLE OF CONTENTS

OBJEKT-ORIENTIERTE PROGRAMMIERUNG

GEBURTSTAGE VON PERSONEN *

Entwickeln Sie ein Programm zur Verwaltung von Personen mit Geburtstagen. Implementieren Sie dazu eine Klasse Person, die Personen mit Name (Vorname und Nachname) und Geburtsdatum darstellt. Für das Datum sollen Sie dabei eine eigene Klasse Date implementieren. Schreiben Sie eine Methode Person readPerson(), die die Daten einer Person einliest, und eine Methode print(), die die Daten einer Person ausgibt.

Schreiben Sie dann eine main-Methode, die Personen von einer Datei einliest und diese im Array speichert. Es sollen dann alle Personen, die an einem bestimmten Jahr, Monat, Tag bzw. Datum (einzulesen von der Konsole) geboren sind, ausgegeben werden.

Schreiben Sie das Programm nach objektorientierten Prinzipien. Das heißt:
- Schreiben Sie Objektmethoden, um die Zugriffe und Aktionen der Klassen Date und Person zu implementieren.
- Machen Sie die Felder private, um den Zugriff auf von außen nicht mehr zu ermöglichen. Greifen Sie dann ausschließlich mit Methoden lesend mit get-Methoden auf die Felder zu.
- Definieren Sie Konstruktoren für die Klassen, um die Objekte bei der Erzeugung sinnvoll zu initialisieren.
- Implementieren Sie die Methoden zum Einlesen der Daten und Erzeugen der Objekte als static Methoden in den Klassen Person und Date (warum müssen diese Methoden static sein?)

MENGENRECHNER *

Für das mathematische Konzept der Menge gibt es in Java kein eingebautes Sprachkonstrukt. Daher muss man Mengen als eine Klasse realisieren. Wir wollen in dieser Aufgabe Mengen für Integer-Zahlen und die üblichen Operationen auf Mengen implementieren. Dabei gehen wir davon aus, dass unsere Mengen ganze Zahlen zwischen 0 und 15 als Elemente enthalten dürfen. Die Elemente einer Menge können in diesem Fall in einem booleschen Array der Länge 16 dargestellt werden, wobei *true* an der Position *i* bedeutet, dass das Element *i* in der Menge enthalten ist, *false* an der Position *i*, dass es nicht enthalten ist.
Beispiel:
Für die Menge M = {2, 5, 6 12} würde sich folgendes boolesches Array ergeben:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
false	false	true	false	false	true	true	false	false	false	false	false	true	false	false	false

Auf Basis dieses Verfahrens sollen Sie eine Klasse *Set* mit folgende Methoden implementieren:

- *Set()* und *Set(int[])*

erzeugen eine Menge, wobei der zweite Konstruktor einen int[]-Parameter mit den Elementen der Menge erwartet

- *static Set readSet()*

erzeugt eine Menge und füllt diese mit von der Konsole eingelesenen int-Werten

- *Set duplicate()*

kopiert eine Menge, d.h. erzeugt ein neues Set-Objekt mit gleichem Inhalt

- *boolean contains(int x)*

prüft ob das Element x in der Menge enthalten ist

- *void union(Set s), void intersect(Set s), void diff(Set s)*

bilden die Vereinigungsmenge, die Schnittmenge und die Mengendifferenz, wobei das Ergebnis in der *this*-Menge gespeichert wird (Objektmethoden)

- *static Set union(Set s1, Set s2), static Set intersect(Set s1, Set s2), static Set diff(Set s1, Set s2)*

bilden ebenfalls die Vereinigungsmenge, die Schnittmenge und die Mengendifferenz, aber als Klassenmethoden (*static*), wobei als Ergebnis jeweils ein neues *Set*-Objekt zurückgegeben wird

Implementieren Sie dann einen Dialog in der Art eines Computer-Mathematiksystems (siehe Beispieldialog). Es sollen Operationen ausgewählt werden können und die Ergebnisse der Operationen unter einem fortlaufenden Index gespeichert werden. Die Ergebnisse der Operationen können dann als Operanden der folgenden Operationen verwendet werden.

Bildschirmdialog:

```
Folgende Operationen stehen zur Verfuegung:
R - Read: Liest eine neue Menge ein
D - Duplicate: Dupliziert eine Menge
U - Union: Vereinigt zwei Mengen
I - Intersect: Schneidet zwei Mengen
F - Diff: Bildet die Differenz von zwei Mengen
M - Member: Prueft ob ein Wert Teil einer Menge ist
. - Exit: Beendet das Programm
```

```
------------------------------------------------------
[0] { }
[1] { 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 }
Kommando (R|D|U|I|F|M|.): R
Elemente (Ende mit -1): 2 4 6 7 9 -1
[2] { 2 4 6 7 9 }
Kommando (R|D|U|I|F|M|.): R
Elemente (Ende mit -1): 4 6 2 12 13 -1
[3] { 2 4 6 12 13 }
Kommando (R|D|U|I|F|M|.): U
Index Operand 1: 2
Index Operand 2: 3
[4] { 2 4 6 7 9 12 13 }
Kommando (R|D|U|I|F|M|.): I
Index Operand 1: 2
Index Operand 2: 3
[5] { 2 4 6 }
Kommando (R|D|U|I|F|M|.): F
Index Operand 1: 2
Index Operand 2: 3
[6] { 7 9 }
Kommando (R|D|U|I|F|M|.): D
Index Operand 1: 5
[7] { 2 4 6 }
Kommando (R|D|U|I|F|M|.): M
Index Operand 1: 7
Wert: 4
true
Kommando (R|D|U|I|F|M|.): M
Index Operand 1: 7
Wert: 1
false
Kommando (R|D|U|I|F|M|.): .
```

PROJEKT

GRUNDANFORDERUNGEN: LAGER FÜR AUTOTEILE

Achte auf die richtige Verwendung von **`final, public, private`!**

Schreiben Sie ein Programm *AutoPartWarehouse*, mit dem man Autoteile verwalten kann. Ein Autoteil (Klasse *Part*) besteht aus einer Artikelnummer, einer Beschreibung und dem Lagerbestand in Stück. Lesen Sie die Autoteile aus der Vorgabedatei *Autoparts.txt*. Die Vorgabedatei hat folgendes Format: am Beginn steht die Anzahl der Autoteile, gefolgt von einer Zeile pro Autoteil mit der Artikelnummer, der Beschreibung und dem Lagerbestand.

Hinweis: die Artikel sind von 0 beginnend fortlaufend nummeriert, d. h. Sie können die Artikelnummer zum Indizieren im Array verwenden.

```
3
0 "Gonf Rapid Engine" 12
1 "F16 Gearbox" 3
2 "Windshield Viper Clear View" 19
```

Speichern Sie die Arrays für Artikel als Objektfeld der Klasse *AutoPartWarehouse*, führen Sie Konstrukturen ein, implementieren Sie statische *read*-Methoden in den passenden Klassen, und implementieren Sie Objektmethoden, z. B. *print()* um ein Autoteil auzugeben.

Testen Sie Ihre Methoden mit folgender *main*-Methode:

```
public class AutoPartProject {
   public static void main(String[] args) {
     AutoPartWarehouse warehouse = new AutoPartWarehouse("Autoparts.txt");
     warehouse.print();
   }
}
```

Vorgabe zu AutoPartWarehouse:

```
public class AutoPartWarehouse {
   private final Part[] parts;
   // TODO
}
```

Die Ausgabe von *print* soll wie folgt aussehen:

```
Part no. | Description                  | Stock
-------------------------------------------------
       0 | Gonf Rapid Engine            |    12
       1 | F16 Gearbox                  |     3
       2 | Windshield Viper Clear View  |    19
```

GRUNDANFORDERUNGEN: BESTELLUNG

Erweitern Sie das vorherige Programm und lesen Sie Bestellungen aus der Vorgabedatei *Orders.txt* ein. Eine Bestellung (Klasse *Order*) besteht aus mehreren Positionen. Eine Position (Klasse *Item*) besteht aus einem Artikel (in der Datei steht die Artikelnummer, im Objekt speichern Sie eine Referenz auf den Artikel) und einer Bestellmenge. Speichern Sie Bestellungen in einem Array in AutoPartWarehouse. Eine Bestellung besteht (nur) aus einem Array von Positionen. Die Vorgabedatei hat folgendes Format: am Beginn steht die Anzahl der Bestellungen, gefolgt von jeder Bestellung in einer eigenen Zeile. In jeder Zeile steht am Anfang die Anzahl der Positionen, gefolgt von Artikelnummer und Bestellmenge pro Position, durch Komma getrennt.

```
2
3 0,1 1,3 2,2
2 1,9 2,2
```

Testen Sie Ihre Methoden mit folgender *main*-Methode:

```
public class AutoPartProject {
   public static void main(String[] args) {
     AutoPartWarehouse warehouse
             = new AutoPartWarehouse("Autoparts.txt", "Orders.txt");
     warehouse.print();
   }
}
```

Vorgabe zu AutoPartWarehouse:

```
public class AutoPartWarehouse {
   private final Part[] parts;
   private final Orders[] orders;
   // TODO
}
```

DARÜBER HINAUSGEHENDE ANFORDERUNGEN: AUFTRAG ABARBEITEN

Erweitern Sie das Programm so, dass alle Aufträge, die erfüllbar sind, abgearbeitet werden. Schreiben Sie dazu in der Klasse *Order* folgende Methoden: *boolean isFulfillable()* prüft, ob der Auftrag mit dem aktuellen Lagerbestand durchführbar ist, d. h. dass von jedem Artikel ausreichend Stück vorhanden sind; *void process()* bucht die Artikel eines Auftrags aus dem Lager aus, d. h. die Lagerbestände werden reduziert.

Testen Sie Ihr Programm mit folgender *main*- und *processOrders*-Methode:

```
public static void main(String[] args) {
  AutoPartWarehouse warehouse = new AutoPartWarehouse("Autoparts.txt", "Orders.txt");
  warehouse.print();
  warehouse.processOrders();
  Out.println("*** Warehouse after processing orders ***\n");
  warehouse.print();
}
```

Die Methode processOrders in der Klasse AutoPartWarehouse soll wie folgt aussehen:

```
void processOrders() {
  for (int i = 0; i < orders.length; i++) {
    Order order = orders[i];
    if (order.isFulfillable()) {
      order.process();
      orders[i] = null;
    }
  }
}
```

Hinweis: Die Methode *processOrders* setzt erfüllte Aufträge im Auftrag-Array auf *null*. Passen Sie Ihre Druckmethode für Aufträge so an, dass *null*-Werte einfach übersprungen werden.

Die Ausgabe soll wie folgt aussehen:

```
PARTS
Part no. | Description                    | Stock
-----------------------------------------------
       0 | Gulf Stream TDI Engine         |    12
       1 | F16 Gearbox                    |     3
       2 | Windshield Viper Eisbiber      |    19

ORDERS
Part no. | Description                    | Qty.
-----------------------------------------------
       0 | Gulf Stream TDI Engine         |     1
       1 | F16 Gearbox                    |     3
       2 | Windshield Viper Eisbiber      |     2
-----------------------------------------------
       1 | F16 Gearbox                    |     9
       2 | Windshield Viper Eisbiber      |     2
-----------------------------------------------

*** Warehouse after processing orders ***

PARTS
Part no. | Description                    | Stock
-----------------------------------------------
       0 | Gulf Stream TDI Engine         |    11
       1 | F16 Gearbox                    |     0
       2 | Windshield Viper Eisbiber      |    17

ORDERS
Part no. | Description                    | Qty.
-----------------------------------------------
       1 | F16 Gearbox                    |     9
       2 | Windshield Viper Eisbiber      |     2
-----------------------------------------------
```

LÖSUNGEN

GEBURTSTAGE VON PERSONEN – OBJEKTORIENTIERT

```
class Person {

     public static Person readPerson() {
          return new Person(In.readString(), In.readString(), Date.readDate());
     }

     private String firstName;
     private String lastName;
     private Date dateOfBirth;

     public Person(String firstName, String lastName, Date dateOfBirth) {
          this.firstName = firstName;
          this.lastName = lastName;
          this.dateOfBirth = dateOfBirth;
     }

     public String getFirstName() {
          return firstName;
     }

     public String getLastName() {
          return lastName;
     }

     public Date getDateOfBirth() {
          return dateOfBirth;
     }

     public void print() {
          Out.print(String.format("%-15s %-15s", firstName, lastName));
          dateOfBirth.print();
     }

     public boolean equals(Person other) {
          return this.firstName.equals(other.firstName)
                    && this.lastName.equals(other.lastName)
```

```
                && this.dateOfBirth.equals(other.dateOfBirth);
    }

    public boolean isBornAt(Date d) {
        return (d.getYear() == 0 || getDateOfBirth().getYear() == d.getYear())
                && (d.getMonth() == 0 || getDateOfBirth().getMonth() == d.getMonth())
                && (d.getDay() == 0 || getDateOfBirth().getDay() == d.getDay());
    }

}

class Date {

    public static Date readDate() {
        return new Date(In.readInt(), In.readInt(), In.readInt());
    }

    private int year;
    private int month;
    private int day;

    Date(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }

    public int getYear() {
        return year;
    }

    public int getMonth() {
        return month;
    }

    public int getDay() {
        return day;
    }

    public void print() {
        Out.print(String.format("%02d.%02d.%04d", day, month, year));
    }

    public boolean equals(Date other) {
        return this.year == other.year && this.month == other.month
                && this.day == other.day;
    }

}
```

```
public class BirthdayFinderOOP {

     public static void main(String[] args) {
          Person[] persons = readPersons("Persons.txt");
          printPersons(persons);
          Out.println();

          Out.print("Filter (Jahr Monat Tag): ");
          Date filter = Date.readDate();
          printPersons(persons, filter);
          Out.println();

     }

     static Person[] readPersons(String filename) {
          In.open(filename);
          if (!In.done()) {
               return new Person[0];
          }
          int personCount = In.readInt();
          Person[] persons = new Person[personCount];
          for (int i = 0; i < personCount; i++) {
               persons[i] = Person.readPerson();
          }
          In.close();
          return persons;
     }

     static void printPersons(Person[] persons) {


          for (int i = 0; i < persons.length; i++) {
               persons[i].print();
               Out.println();
          }
     }

     static void printPersons(Person[] persons, Date filter) {
          for (int i = 0; i < persons.length; i++) {
               Person p = persons[i];
               if (p.isBornAt(filter)) {
                    p.print();
                    Out.println();
               }
          }
     }

}
```

MENGENRECHNER

Set.java

```
1  class Set {
2
3    static final int NUM_VALUES = 16;
4
5    private boolean[] values;
6
7    Set() {
8      values = new boolean[NUM_VALUES];
9    }
10
11   Set(int[] init) {
12     this();
13     for (int i = 0; i < init.length; i++) {
14       values[init[i]] = true;
15     }
16   }
17
18   static Set createEmpty() {
19     return new Set();
20   }
21
22   static Set createFull() {
23     return new Set(new int[] {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15});
24   }
25
26   static Set readSet() {
27     Out.print("Elemente (Ende mit -1): ");
28     Set set = new Set();
29     int val = 0;
```

```
30      do {
31        val = In.readInt();
32        if (In.done() && val != -1) {
33          if(val >= 0 && val < NUM_VALUES) {
34            set.values[val] = true;
35          }
36          else {
37            Out.println("Ungueltiger Wert");
38          }
39        }
40      } while (val != -1);
41      return set;
42    }
43
44    String asString() {
45      StringBuilder b = new StringBuilder();
46      b.append("{");
47      for (int i = 0; i < values.length; i++) {
48        if (values[i]) {
49          b.append(' ');
50          b.append(i);
51        }
52      }
53      b.append(" }");
54      return b.toString();
55    }
56
57    Set duplicate() {
58      Set clone = new Set();
59      System.arraycopy(values, 0, clone.values, 0, values.length);
60      return clone;
61    }
62
63    boolean contains(int x) {
64      return x >= 0 && x < values.length && values[x];
65    }
66
67    void union(Set s) {
68      for (int i = 0; i < values.length; i++) {
69        values[i] |= s.values[i];
70      }
71    }
72
73     static Set union(Set set1, Set set2) {
74       Set result = set1.duplicate();
75       result.union(set2);
76       return result;
77     }
78
79     void intersect(Set s) {
80       for (int i = 0; i < values.length; i++) {
81         values[i] &= s.values[i];
82       }
83     }
84
85     static Set intersect(Set set1, Set set2) {
86       Set result = set1.duplicate();
87       result.intersect(set2);
88       return result;
89     }
90
91     void diff(Set s) {
92       for(int i=0; i < values.length; i++) {
93         values[i] &= !s.values[i];
94       }
95     }
96
97     static Set diff(Set set1, Set set2) {
98       Set result = set1.duplicate();
99       result.diff(set2);
100      return result;
101    }
102  }
```

SetCalculator.java

```
1  import java.util.Arrays;
2
3  class SetCalculator {
4
5    static int cnt = 0;
6    static Set[] sets = new Set[1];
7
8    public static void main(String[] args) {
9      printHelp();
10
11     addSet(Set.createEmpty());
12     addSet(Set.createFull());
13     printSet(0);
14     printSet(1);
15
16     while (true) {
17       Set set = null;
18       switch (readCommand()) {
19         case 'r':
20           set = Set.readSet();
21           break;
22         case 'd':
23           set = sets[readOperand(1)].duplicate();
24           break;
25         case 'u':
26           set = Set.union(sets[readOperand(1)], sets[readOperand(2)]);
27           break;
28         case 'i':
29           set = Set.intersect(sets[readOperand(1)], sets[readOperand(2)]);
30           break;
31         case 'f':
32           set = Set.diff(sets[readOperand(1)], sets[readOperand(2)]);
33           break;
34         case 'm':
35           Out.println(sets[readOperand(1)].contains(readIndex()));
36           break;
37         case '.':
38           return;
39       }
40       if (set != null) {
41         addSet(set);
42         printSet(cnt-1);
43       }
44     }
45   }
46
47   static void printHelp() {
48     Out.println("Folgende Operationen stehen zur Verfuegung:"
49       +"\nR - Read: Liest eine neue Menge ein"
50       +"\nD - Duplicate: Dupliziert eine Menge"
51       +"\nU - Union: Vereinigt zwei Mengen"
52       +"\nI - Intersect: Schneidet zwei Mengen"
53       +"\nF - Diff: Bildet die Differenz von zwei Mengen"
54       +"\nM - Member: Prueft ob ein Wert Teil einer Menge ist"
55       +"\n. - Exit: Beendet das Programm"
56       +"\n------------------------------------------------------");
57   }
58
59   static char readCommand() {
60     Out.print("Kommando (R|D|U|I|F|M|.): ");
61     char ch = ' ';
62     while (-1 == ".rduifm".indexOf(ch)) {
63       ch = Character.toLowerCase(In.readChar());
64     }
65     return ch;
66   }
67
68   static int readOperand(int num) {
69     return readValue(String.format("Index Operand %d: ", num), 0, cnt-1);
70   }
71
```

```
72    static int readIndex() {
73      return readValue("Wert: ", 0, Set.NUM_VALUES);
74    }
75
76    static int readValue(String message, int min, int max) {
77      while (true) {
78        Out.print(message);
79        int index = In.readInt();
80        if(index >= min && index <= max) {
81          return index;
82        }
83        Out.println("Ungueltige Eingabe");
84      }
85    }
86
87    static void addSet(Set set) {
88      if (cnt == sets.length) {
89        sets = Arrays.copyOf(sets, sets.length * 2);
90      }
91      sets[cnt] = set;
92      cnt++;
93    }
94
95    static void printSet(int pos) {
96      Out.println(String.format("[%d] %s", pos, sets[pos].asString()));
97    }
98
99  }
```

```
Folgende Operationen stehen zur Verfuegung:
R - Read: Liest eine neue Menge ein
D - Duplicate: Dupliziert eine Menge
U - Union: Vereinigt zwei Mengen
I - Intersect: Schneidet zwei Mengen
F - Diff: Bildet die Differenz von zwei Mengen
M - Member: Prueft ob ein Wert Teil einer Menge ist
. - Exit: Beendet das Programm
-------------------------------------------------------
[0] { }
[1] { 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 }
Kommando (R|D|U|I|F|M|.): R
Elemente (Ende mit -1): 2 4 6 7 9 -1
[2] { 2 4 6 7 9 }
Kommando (R|D|U|I|F|M|.): R
Elemente (Ende mit -1): 4 6 2 12 13 -1
[3] { 2 4 6 12 13 }
Kommando (R|D|U|I|F|M|.): U
Index Operand 1: 2
Index Operand 2: 3
[4] { 2 4 6 7 9 12 13 }
Kommando (R|D|U|I|F|M|.): I
Index Operand 1: 2
Index Operand 2: 3
[5] { 2 4 6 }
Kommando (R|D|U|I|F|M|.): F
Index Operand 1: 2
Index Operand 2: 3
[6] { 7 9 }
Kommando (R|D|U|I|F|M|.): D
Index Operand 1: 5
[7] { 2 4 6 }
Kommando (R|D|U|I|F|M|.): M
Index Operand 1: 7
Wert: 4
true
Kommando (R|D|U|I|F|M|.): M
Index Operand 1: 7
Wert: 1
false
Kommando (R|D|U|I|F|M|.): .
```