**Aufgabe 1** (Modellieren und Schreiben von Klassen)

Sie sollen in dieser Aufgabe Klassen entwickeln, die es erlauben, Informationen über Prüfungsordnungen zu speichern.

a) Entwickeln Sie eine Klasse `PO`. Ein Objekt dieser Klasse repräsentiert eine Prüfungsordnung eines Studiengangs. Es soll drei Eigenschaften haben: einen Namen des Studiengangs, der durch eine Zeichenkette dargestellt ist, ein Aufzählungsobjekt, der angibt, ob es sich um ein Bachelor- oder Masterstudiengang handelt und das Jahr, in dem die Prüfungsordnung in Kraft getreten ist.

Sie benötigen dabei auch die Definition einer Aufzählungsklasse.

Schreiben Sie einen Konstruktor, der alle Felder initialisiert.

Überschreiben Sie die Methode `equals` auf adequate Weise.

Überschreiben Sie die Methode `toString`.

b) Schreiben Sie eine Unterklasse `Dual` der Klasse `PO`, die eine Prüfungsordnung dualer Studiengänge reprüsentiert. Ein dualer Studiengang habe zusätzlich eine Angabe über die Anzahl der in einem Betrieb abzuleistenden Stunden.

c) Gegeben Sei die Klasse `Fachbereich`, die eine Liste von Studiengängen enthält:

```
1 ..... Fachbereich{
2   java.util.List<PO> pos = new java.util.ArrayList<>();
3 }
```

Listing 1: Fachbereich.java

Schreiben Sie für diese Klasse eine Methode

```
int anzahlDerDualenStudiengänge()
```

die die Anzahl der der dualen Studiengänge in der Liste zurück gibt.

d) Schreiben Sie für diese Klasse eine Methode

```
PO ältestePO(),
```

die die am längsten gültige Prüfungsordnung berechnet.

Sie dürfen davon ausgehen, dass die Liste nicht leer ist.

```
enum Typ {
  bachelor, master; }
class PO {
  String name;
  Typ typ;
  int jahr;
  PO(String n, Typ t, int j){
    name = n; typ = t; jahr = j;
  }

  @Override
  public boolean equals(Object o){
    if(o == null) return false;
    if(!(o instanceof PO)) return false;
    var that = (PO) o;
    return this.name.equals(that.name)
      && this.typ == that.typ
      && this.jahr == that.jahr;
  }
  @Override
  public String toString(){
    return typ + " " + name + " " + jahr;
  }
}
```

b)
```
class Dual extends PO {
  int stunden;
  Dual(String n, Tup t, int j, int s){
    super(n, t, j);
    stunden = s;
  }
}
```

c)
```
int anzahlDualerStudiengänge(){
  var r = 0;
  for(var po : pos){
    if(po instanceof Dual) r++;
  }
  return r;
}
```

d)
```
PO älteresPO(){
  var r = pos.get(0);
  for(var po : pos){
    if(po.jahr < r.jahr) r = po;
  }
  return r;
}
```

Name: Matrikelnummer:

**Aufgabe 2** (Programmfluss)

a) Führen Sie die folgende Klasse von Hand aus. Schreiben Sie dabei auf, in welcher Reihenfolge die Zeilen durchlaufen werden und mit welchen Werten die einzelnen Variablen während des Programmdurchlaufs belegt sind. Schreiben Sie auf, was auf dem Bildschirm ausgegeben wird.

```
1  class Aufgabe2a{
2    public static void main(String[] args){
3      int y = 4;
4      for (int x = 25;!(y>x-1);x--){
5        switch (x-1){
6          case 20:;
7          case 8:;
8          case 42:;
9          case 9: x=x-y-2;
10         case 11: y--;break;
11         case 1: y++;return;
12         default: x--;
13       }
14       x -=1;
15       System.out.println(x+   +y);
16       x--;
17     }
18     System.out.println(y);
19   }
20 }
```

Listing 2: Aufgabe2a.java

| Zeilen | x | y | Ausgabe |
|---|---|---|---|
| 3, 4 | 25 | 4 | |
| 12 | 24 | | |
| 14, 15 | 23 | | "23 4" |
| 16 | 22 | | |
| 4 | 21 | | |
| 5, 11 | 15 | 3 | |
| 14, 15 | 14 | | "14 5" |
| 16 | 13 | | |
| 4, 10 | 12 | 2 | "11 2" |
| 14, 15 | 11 | | |
| 16 | 10 | | |
| 4 | 9 | | |
| 5, 10 | 5 | 1 | "4 1" |
| 14, 15 | 4 | | |
| 16, 4, 11 | 3, 2 | 2 | |

b) Führen Sie die folgende Klasse von Hand aus. Schreiben Sie dabei auf, in welcher Reihenfolge die Zeilen durchlaufen werden und mit welchen Werten die einzelnen Variablen während des Programmdurchlaufs belegt sind. Schreiben Sie auf, was auf dem Bildschirm ausgegeben wird.

```
1  ... Aufgabe2b{
2    ... ... ... main(String [] args){
3      ... [] xs = {5,13};
4      ... y = 21;
5
6      ... (var x:xs){
7        ... (... i=y;i>0;i=i-6){
8          System.out.println(    +i+   +x+   +y);
9          x += 3;
10         y--;
11         ... (y==19){
12           y=y+1;
13           ...;
14         }
15       }
16       ... (x == 13){
17         x=15;
18         ...;
19       }
20       x--;
21       System.out.println(    +x+   +y);
22     }
23   }
24 }
```

Listing 3: Aufgabe2b.java

| Zeilen | x | y | i | Ausgabe |
|---|---|---|---|---|
| 4,6,7,8 | 5 | 21 | 21 | "I 21 5 21" |
| 9,10 | 8 | 20 | | |
| 7,9,10 | 11 | 19 | 15 | "I 15 19" |
| 12,20,21 | 10 | 20 | | "A 10 20" |
| 6 | 13 | | 20 | "I 20 13 20" |
| 9,10 | 16 | 19 | | |
| 12,20 | 15 | 20 | | "A 15 20" |

✓

c) Betrachten Sie folgende Klasse:

```
class Aufgabe2c{
  static int f(int a) {
    return a == 0 ? 0 : (f(a/2)+a%2) ;
  }

  public static void main(String [] args){
    System.out.println(f(27));
  }
}
```

Listing 4: Aufgabe2c.java

Berechnen Sie schrittweise das Ergebnis des Ausdrucks `f(27)`.

f(27)

→ f(13) + 1

→ f(6) + 1 + 1

→ f(3) + 0 + 1 + 1

→ f(1) + 1 + 0 + 1 + 1

→ f(0) + 1 + 1 + 0 + 1 + 1

→ 0 + 1 + 1 + 0 + 1 + 1

→ 4

d) Betrachten Sie folgende Klassen:

```
class A2dO{
  int f1(int x){
    return f2(x);
  }
  int f2(int x){
    return x+42;
  }
}
```

Listing 5: A2dO.java

```
class A2dU extends A2dO{
  int f2(int x){
    return x+17;
  }

  int f1(int x){
    return x-17;
  }

  public static void main(String[] args){
    A2dO x1 = new A2dO();
    A2dO x2 = new A2dU();
    System.out.println(x1.f1(2));
    System.out.println(x2.f1(2));
    System.out.println(x1.f2(2));
    System.out.println(x2.f2(2));
  }
}
```

Listing 6: A2dU.java

Welche Ausgabe tätigt die Klasse `A2dU` bei Ausführung auf der Kommandozeile. Erklären Sie, wie es zu dieser Ausgabe kommt.

13: 2+42 = 44
14: 2-17 = -15
15: 2+42 = 44
16: 2+17 = 19
Hier sieht man das Late-Binding

**Aufgabe 3** (Ausdrücke und Anweisungen)

Gegeben sei eine einfache Klasse für Längenangaben im angloamerikanisches Maßsystem. 12 Inch sind dabei ein Foot.

```
1  public class Distance{
2    private int foot;
3    private int inch;
4    public Distance(int newFoot, int newInch){
5      foot = newFoot;
6      inch = newInch;
7    }
8
9    public String toString(){
10     return (foot+" "+inch);
11   }
12 }
```

Listing 7: Distance.java

a) Schreiben Sie für die Klasse `Distance` eine Methode `isShorterThan`, die angibt, dass das Objekt eine kürzere Entfernung beschreibt, als der übergebene Parameter. Verwenden Sie dabei außer einem `return` keine weitere Anweisung und verwenden Sie nicht den Bedingungsoperator.

```
13   public boolean isShorterThan(Distance that){
14     return
15       this.foot < that.foot
16       || (this.foot == that.foot
17         && this.inch < that.inch);
18   }
```

Listing 8: Distance.java

b) Schreiben Sie für die Klasse eine Methode, die anzeigt, dass das this-Objekt eine längere Distanz als `that` beschreibt:

Verwenden Sie hierbei die Methode aus Aufgabenteil a).

```
29        isLongerThan(Distance that){
30
31          return that.isShorterThan(this);
32
33
34
35
36
37
38
39
40    }
```

Listing 9: Distance.java

c) Schreiben Sie für die Klasse `Distance` eine Methode `addInch`, die das Objekt modifiziert, so dass es die als Parameter übergebene Länge in Inch länger ist:

```
41    void addInch( ... inch){
42
43      var i = this.inch + inch;
44
45
46      this.Foot += i/12;
47
48      this.inch = i % 12;
49
50
51
52
53
54    }
```

Listing 10: Distance.java

d) Ein Inch ist die Länge von 25,4 Millimeter. Schreiben Sie für die Klasse `Distance` eine Methode, die die beschriebene Länge in Millimetern berechnet:

```
35  int getDistanceInMillimeter(){
36      return (foot *12 + inch) * 25.4;
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51  }
```

Listing 11: Distance.java

a) Schreiben Sie in der Klasse `AL` eine Methode `firstIndexOf`:

Es soll der erste Index aus der Liste berechnet werden, an dem ein Objekt gespeichert ist, das gleich zum Parameter `o` ist. Ist kein solches Objekt enthalten, so soll -1 zurück gegeben werden.

```
...  firstIndexOf(A o){
     for (int i=0; i<size(); i++){
       if(o.equals(get(i)))
         return i;
     }
     return -1;
}
```

Listing 13: AL.java

**b)** Gegeben sei zusätzlich folgende Schnittstelle:

```
1 [illegible] Property<A>{
2   [illegible] test(A a1);
3 }
```

Listing 14: Property.java

Sie soll verwendet werden, um zu testen, ob ein Objekt eine bestimmte Eigenschaft hat.

Schreiben Sie jetzt in der Klasse `AL` eine Methode `firstIndexOf(Property<A> p)`.

Es soll der erste Index aus der Liste berechnet werden, an dem ein Objekt gespeichert ist, das die Eigenschaft `p` hat. Ist kein solches Objekt enthalten, so soll -1 zurück gegeben werden.

```
41   firstIndexOf(Property<A> p){
42   for(int i=0; i<size(); i++){
43     if(p.test(get(i)))
44       return i;
45   }
46   return -1;
47 }
```

Listing 15: AL.java

**Aufgabe 4 (Arraylisten)**

Gegeben sei folgende Klasse, die eine einfache Array-basierte Liste realisiert, wie aus der Vorlesung bekannt.

```
public class AL<A> {
  private int size = 0;
  private A[] array = (A[]) new Object[10];

  public void add(A el){
    if (size >= array.length){
      enlargeArray();
    }
    array[size++] = el;
  }
  private void enlargeArray(){
    A[] newarray = (A[]) new Object[array.length + 10];
    for (int i=0; i<array.length; i++) newarray[i] = array[i];
    array = newarray;
  }
  public A get(int i) {
    return array[i];
  }
  public int size(){return size;}
  public boolean isEmpty(){return size == 0;}
}
```

Listing 12: AL.java

Implementieren Sie folgende Methoden für diese Listenklasse:

c) Schreiben Sie jetzt die Methode aus Aufgebenteil a) neu, unter Verwendung der Methode von Aufgabenteil b).

```
... firstIndexOf(A o){
  return firstIndexOf(x -> o.equals(x));
}
```

Property

Listing 16: AL.java

d) Schreiben Sie folgende Methode `deleteAt(int i)`. Die Liste soll so verändert werden, dass das Element am Index `i` aus der Liste gelöscht wird und die Liste anschließend ein Element weniger hat. Ist der Index `i` kein gültiger Index für ein Element, so bleibt die Liste unverändert.

```
... deleteAt(... i){
  if(i<0 || i >= size()) return;
  for( ; i < size()-1; i++){
    array[i] = array[i+1];
  }
  size--;
}
```

Listing 17: AL.java

**Aufgabe 5** Zusammenhänge

Erklären Sie in kurzen Worten.

a) Betrachten Sie folgende Methode:

```
1 static int f(int x){
2   while (x<0){
3     x *= -1;
4   }
5   return x;
6 }
```

Mit welcher sonst so verpönten Bezeichnung kann die Schleife zu Recht benannt werden. Formulieren Sie die Methode um, so dass keine Schleife verwendet wird.

Dieses ist quasi eine if-schleife, denn sie wird maximal einmal durchlaufen.
Das Schlüsselwort while kann hier durch if ersetzt werden.

b) Was gilt für alle abstrakten Methoden einer Schnittstelle und ist bei der Implementierung einer Schnittstelle zu beachten.

Methoden aus Schnittstellen sind immer public, auch wenn das in der Schnittstelle nicht durch das Attribut vermerkt ist.

c) Welche Sichtbarkeiten für Methoden hat Java.

public
private
protected
sind die Attribute für Sichtbarkeiten.
Ein Attribut ist eine Methode
package Sichtbar.

d) Geben Sie drei Beispiele für statische Analysen, die der Javacompiler durchführt.

- Typcheck
- checked Exceptions nicht gefangen oder im throws declariert.
- check ob alle Methoden des implementierten Interfaces auch implementiert sind.
- missing return in Methode
- ob Sichtbarkeit verletzt wurde
- ob Final verletzt wird

e) Erklären Sie, welche Funktion Konstruktoren in Aufzählungsklassen haben.

Dieser wird genutzt, um jeder der endlich vielen Objekte der Aufzählung Feldern zu initialisieren.
Der Konstruktor einer Aufzählungsklasse ist immer privat.