Aufgaben zur Veranstaltung
Programmierung mit Java, WS 2024/2025

H. Pflug, J. Dietel, J. Fuchs FH Aachen, Campus Jülich; IT Center, RWTH Aachen

Klausurvorbereitung Blatt 4 22./23.01.2025

1.) a) Wie heißt der Rückgabetyp, der angegeben wird, wenn kein Wert zurückgegeben wird?

b) Wie heißt das Schlüsselwort für „kein Objekt“?

c) Wie heißt die Klasse, die alle anderen Klassen als Basisklasse haben?

d) Welche Basisklasse müssen *unchecked Exceptions* in Java haben?

e) Wenn *Bruch* die Basisklasse zu *XBruch* ist, dann ist *XBruch* zu *Bruch* die ...?

f) Welche Bindungsart wird in Java für „normale“ Methoden einer Klasse benutzt?

g) Was ist der Oberbegriff, unter dem die Klassen `Integer`, `Double` und `Character` bekannt sind?

h) Welcher Mechanismus tritt im Hintergrund bei der Zeile

```
Integer x = 5;
```

in Kraft?

i) Lösungsschablonen für wiederkehrende Probleme aus der Software-Entwicklung kennt man unter dem folgenden Namen:

j) Nennen Sie zwei wichtige javadoc-Tags.

k) Was bedeutet das Schlüsselwort `final`?

l) Was ist der Unterschied zwischen `double` und `Double`?

m) Mit welchem Schlüsselwort kann man auf Methoden der Basisklasse zugreifen?

n) Nennen Sie einen Fall, in dem Java die statische Bindung benutzt.

o) Nennen Sie einen Nachteil von abstrakten Klassen gegenüber Interfaces.

p) Was bewirkt ein Aufruf der folgenden Funktion?

```
public static void swap(int x, int y) {
  int temp = x;
  x = y;
  y = temp;
}
```

q) Wie lautet das Schlüsselwort, das man benötigt, um eine Exception auszulösen?

2.) Gegeben ist ein Interface

```
public interface Mathfunktion {
  public double f(double x);
}
```

Schreiben Sie den Code, der nötig ist, um die Variable

```
Mathfunktion x;
```

mit einem Objekt zu belegen, das die Funktion $f(x) = x^2$ realisiert.

3.) Welches Prinzip der Objektorientierung wird mit folgendem Code verletzt? Verbessern Sie den Code.

```
public class Feld {
    private ArrayList<Integer> x;

    public Feld(ArrayList<Integer> x) {
        this.x=x;
    }
}
```

4.) Die folgenden Programmausschnitte enthalten insgesamt **genau zwei** Compiler-Fehler. Finden Sie diese und erläutern Sie je Fehler kurz die Ursache.

```
class C {
   public void f() { }
   public void g() { super.g(); }
   public String toString() { return super.toString(); }
}

class D extends C {
   public void f() { super.f(); }
   public void g() { }
}
```

```
C c = new C();
D d = new D();

c = d;
d = c;
```

5.) Die Primzahleigenschaft einer natürlichen Zahl z kann durch Ausprobieren aller potentiellen Teiler von 2 bis z-1 überprüft werden: ist keiner dieser potentiellen Teiler ein echter Teiler von z, dann ist z eine Primzahl. Diesen Brute-Force-Primzahltest kann man mit einer for-Schleife implementieren. Es geht aber auch rekursiv.

Die Funktion `istPrimzahl(int p)` sei wie folgt mit Hilfe der rekursiven Funktion `istPrimzahl(int p, int z)` definiert:

* `istPrimzahl(p) := istPrimzahl(p, p-1)`
* `istPrimzahl(p, 1) := true`
* `istPrimzahl(p, z) := false, falls p durch z teilbar ist (z>1)`
* `istPrimzahl(p, z) := istPrimzahl(p, z - 1), falls p nicht durch z teilbar ist (z>1)`

Implementieren Sie die Funktion `istPrimzahl(int p)` und die Hilfsfunktion rekursiv nach dem oben beschriebenen Schema. Werte von $p \leq 0$ lösen eine Exception aus. Beachten Sie auch den Fall $p = 1$.

6.) Schreiben Sie eine Funktion

```
public static int groessterSingulaererWert(int[] arr)
```

die von allen Werten des Feldes `arr`, die dort nur ein einziges Mal (also nicht mehrfach) vorkommen, den größten zurückgibt. Sollte in `arr` kein singulärer Wert vorhanden sein, wird eine `ArithmeticException` ausgelöst.

7.) Schreiben Sie eine Funktion

```
public boolean testeKlammerung(String kl)
```

Der String *kl* enthält eine Zeichenkette mit runden Klammern. Es soll überprüft werden, ob die enthaltenen runden Klammern den Regeln einer vollständigen Klammerung entsprechen. Das heißt, jede sich öffnende Klammer muss auch wieder geschlossen werden und jede sich schließende Klammer muss zuvor geöffnet worden sein. Die Regeln sind auch dann erfüllt, wenn im String gar keine runde Klammer auftaucht. Falls *kl* den Regeln entspricht, wird `true` zurückgegeben, ansonsten `false`.

8.) Die Iterationsvorschrift zur Berechnung eines Apfelmännchens ist:

$$x_0 = 0; \quad y_0 = 0$$

$$x_{n+1} = x_n^2 - y_n^2 + c_x$$

$$y_{n+1} = 2x_n y_n + c_y$$

Schreiben Sie eine Methode

```
public int zaehleIterationen(double cx, double cy)
```

die zählt, wie viele Iterationen nötig sind, damit die Gleichung

$$x_n^2 + y_n^2 > 4$$

erfüllt ist. Ist nach 200 Iterationen die Gleichung noch nicht erfüllt, wird die Iteration abgebrochen und der Wert 200 zurückgegeben.

9.) In Java gibt es die Klasse *java.util.Random*, mit der sich eine Folge von Zufallszahlen des Typs *int* erzeugen lässt, die über den ganzen Wertebereich von *int* gleichmäßig verteilt ist. Der folgende Code gibt 2 *int*-Zufallszahlen aus:

```
Random r = new Random();
System.out.println(r.nextInt());
System.out.println(r.nextInt());
```

Schreiben Sie eine Methode

```
public int ersteDoppelte(Random r)
```

die zurückgibt, wieviele Zufallszahlen erzeugt werden konnten, bevor die erste Zufallszahl doppelt auftrat.

Hinweis: Benutzen Sie für Ihre Lösung die Klasse *ArrayList*, die Klasse *HashMap* oder eine ähnliche Klasse.

10.) **Menge.** Schreiben Sie eine Klasse `Menge`. Die Klasse enthält eine Menge an Integer-Zahlen. Jede Integer-Zahl darf nur einmal in der Menge vorhanden sein. Insgesamt kann die Menge aber beliebig groß werden. Die Klasse soll folgende Konstruktoren und Methoden enthalten:

```
public Menge();                  //Erzeugt eine leere Menge
public void add(int x);          //Fuegt das Element x ein,
                                 //falls es noch nicht vorhanden ist.
public int size();               //Gibt die Anzahl der Elemente zurueck
public boolean contains(int x);  //Gibt zurueck, ob x in der Menge enthalten ist.
public void add(Menge m);        //Nimmt die Elemente aus der Menge m auf.
public Menge getIntersection(Menge m);
                                 //Gibt die Schnittmenge zwischen
                                 //this und m zurueck
```

11.) Schreiben Sie eine Klasse `DoubleSet` („Doppel-Menge“). Die Klasse nimmt `int`-Zahlen auf. Im Gegensatz zu einer normalen Menge dürfen Zahlen auch doppelt vorkommen, jedoch nicht mehr als doppelt. Implementieren Sie die folgenden Methoden:

```
public void add(int z)
```

Fügt die Zahl `z` der Menge hinzu. Ist die Zahl schon zweimal vorhanden, wird eine `ArithmeticException` ausgelöst.

```
public int getCount(int z)
```

Gibt zurück, wie oft die Zahl `z` vorhanden ist. Mögliche Rückgabewerte sind 0, 1 und 2.

12.) Gegeben sei eine Folge, die durch ein gegebenes x_0 und eine Iterationsvorschrift $x_{n+1} = f(x_n)$ definiert ist. Schreiben Sie dazu eine Klasse `Folge` und ein Interface `Iterationsvorschrift`.

a) Entwerfen Sie das Interface `Iterationsvorschrift`.

b) Schreiben Sie die Klasse `Folge` mit folgenden Konstruktoren und Methoden

- `public Folge(double x0, Iterationsvorschrift v)`

  Erzeugt eine Folge mit dem Anfangswert x_0 und der Iterationsvorschrift `v`.

- `public double getNextElement()`

  Gibt beim ersten Aufruf das erste Element der Folge zurück, beim zweiten Aufruf das zweite usw.

- `public void reset()`

  Setzt die Folge zurück, so dass beim nächsten Aufruf von `getNextElement` wieder das erste Element zurückgegeben wird.

c) Schreiben Sie eine Klasse `TestVorschrift`, die das Interface `Iterationsvorschrift` implementiert und die Iterationsvorschrift

$$x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n}$$

darstellt.

13.) Funktionen, die sowohl unendlich oft integrierbar als auch unendlich oft differenzierbar sind, sollen integriert bzw. differenziert werden.

a) Entwerfen Sie ein Interface `DiffIntFunc` für derartige Funktionen mit den folgenden Eigenschaften:

(1) Das Interface umfasst je eine Methode zum Integrieren und Differenzieren.

(2) Dabei gilt: Das Integral einer `DiffIntFunc` ist wiederum eine `DiffIntFunc`. Auch die Ableitung einer `DiffIntFunc` ist wieder eine `DiffIntFunc`.

b) Schreiben Sie eine Klasse `SinCosFunc` für Funktionen, die aus der Addition eines Sinus- und eines Cosinusterms bestehen:

$$f(x) = a \cdot \sin(b \cdot x) + c \cdot \cos(d \cdot x); \quad b, d \neq 0.$$

$f(x)$ ist unendlich oft differenzierbar und integrierbar (das muss nicht geprüft werden). Daher soll `SinCosFunc` das Interface aus Aufgabenteil (a) implementieren.

Hinweise zum Teil (b):

- Stellen Sie sicher, dass die Parameter b und d nie den Wert 0 annehmen. Werfen Sie eine `ArithmeticException`, falls versucht wird, b oder d auf 0 zu setzen. Die Exception-Klasse muss nicht selbst geschrieben werden.
- Getter- und Setter-Methoden für private Attribute können Sie der Kürze halber weglassen.
- $$\begin{aligned} \frac{d}{dx} f(x) &= ab \cdot \cos(b \cdot x) - cd \cdot \sin(d \cdot x) \\ \int f(x)\, dx &= -\frac{a}{b} \cdot \cos(b \cdot x) + \frac{c}{d} \cdot \sin(d \cdot x) \end{aligned}$$
- Integrationskonstanten können missachtet werden!