Check und Prepare

Lösungsvorschläge

TECHNISCHE UNIVERSITÄT DARMSTADT

Autoren: Nhan Huynh und Darya Nikitina
Fachbereich: Informatik
Übungsblatt: 03
Version: 20. November 2021
Semesterübergreifend

Dokumentation in Java: Mittels JavaDoc und den Tags `@param` und `@return` wollen wir nun im Java-Teil eine geeignete Dokumentation unserer Methoden vornehmen. Das JavaDoc können Sie automatisch vor jeder Methode mittels `/**` in einer integrierten Entwicklungsumgebung generieren, wenn Sie danach Enter drücken. Ein Beispiel wäre:

```java
/**
 * This method accepts two real numbers that belong to a
 * vector and calculates the euclidean norm of the said
 * vector.
 *
 * @param x the first component of a two-dimensional vector (x, y)
 * @param y the second component of a two-dimensional vector (x, y)
 * @return the euclidean norm of the vector (x, y)
 */
double euclid2(float x, float y) {
  return Math.sqrt(x*x + y*y);
}
```

V1 Die Würfel sind gefallen!

Mit der Funktion `Math.random()` können Sie eine Zufallszahl im Bereich `0` (inklusive) und `1` (exklusive) erzeugen. Schreiben Sie nun eine Methode `void diceRoll()`.

Diese soll einen Würfelwurf simulieren und die gewürfelte Augenzahl auf der Konsole zurückgeben. Dabei soll der Würfel fair sein, das heißt alle Augenzahlen sollen mit identischer Wahrscheinlichkeit auftreten.

Hinweis Überlegen Sie sich, wie Sie die erzeugten Zahlen aus dem Intervall $[0, 1)$ auf die diskrete Menge $\{1, 2, 3, 4, 5, 6\}$ abbilden können. Mit der Funktion `Math.ceil()` können Sie zur nächst größeren, ganzen Zahl aufrunden.

Übungsblatt 03: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

```
/**
 * Prints a random dice number between 1 and 6 (inclusive).
 */
void diceRoll() {
  // Min + (int) (Math.random() * ((Max - Min) + 1)) = [Min, Max]
  int number = 1 + (int) (Math.random() * ((6 - 1) + 1));
  System.out.println(number);
}
```

V2 Schleifen

```
int x = 0;
while (Math.pow(x, 2) <= 1000) {
  x++;
}
System.out.println(x);
```

(1) Welche Funktion erfüllt der oben stehende Code?

(2) Ersetzen Sie die `while`-Schleife einmal durch eine `for`- und einmal durch eine `do-while`-Schleife, und zwar so, dass weiterhin die gleichen Schritte ausgeführt werden.

Lösungsvorschlag:

(1) Die Methode gibt die nächstgrößere Quadratzahl zu der Zahl `1000` zurück. Also die Zahl `x`, sodass $x^2 > 1000$

(2)

```
int x = 0;
for (; Math.pow(x, 2) <= 1000; ) {
  x++;
}
```

```
int x = -1;
do {
  x++;
} while (Math.pow(x, 2) <= 1000);
```

Information: Der Körper einer `do-while`-Schleife wird immer mindestens einmal ausgeführt, egal ob die Bedingung gültig ist oder nicht. Bei einer `while`-Schleife wird der Körper erst betreten, wenn die Bedingung gültig (also `true`) ist und es kann sein, dass der Körper kein einziges Mal ausgeführt wird.

Übungsblatt 03: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V3 Interfaces

Schreiben Sie ein Interface `I1` mit einer parameterlosen `void`-Methode `m1`. Nun schreiben Sie ein Interface `I2`, das von `I1` erbt. `I2` hat eine zusätzliche `void`-Methode `m2` mit einem `int`-Parameter `i`. Abschließend schreiben Sie eine Klasse `C1`, die das Interface `I2` implementiert. Die Klasse hat ein `int`-Attribut `number`. Beim Aufruf der Methode `m1` soll `number` auf `-1` gesetzt werden. Wird Methode `m2` aufgerufen, so soll `number` auf den übergebenen Wert gesetzt werden.

Lösungsvorschlag:

```
1  public interface I1 {
2
3    void m1();
4
5  }
6
7  public interface I2 extends I1 {
8
9    void m2(int i);
10
11 }
12
13 public class C1 implements I2 {
14
15   int number;
16
17   @Override
18   public void m1() {
19     number = -1;
20   }
21
22   @Override
23   public void m2(int i) {
24     number = i;
25   }
26
27 }
```

Übungsblatt 03: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V4 Geometrische Formen I

Gegeben seien folgende zwei Klassen:

```
1  public class Circle {
2
3    public double radius;
4  }
5
6  public class Rectangle {
7
8    public double length;
9    public double width;
10 }
```

Schreiben Sie zunächst ein Interface `ComputeArea` mit einer parameterlosen `double`-Methode `computeArea`. Erweitern Sie nun die zwei oben genannten Klassen, sodass beide das Interface `ComputeArea` implementieren. Die Methode `computeArea` soll den Flächeninhalt des Kreises bzw. des Rechtecks berechnen und zurückliefern.

Lösungsvorschlag:

```
1  public interface ComputeArea {
2
3    /**
4     * Computes the area.
5     *
6     * @return the area
7     */
8    double computeArea();
9  }
10 public class Circle implements ComputeArea {
11
12   public double radius;
13
14   @Override
15   public double computeArea() {
16     return Math.PI * radius * radius;
17   }
18 }
19 public class Rectangle implements ComputeArea {
20
21   public double length;
22   public double width;
23
24   @Override
25   public double computeArea() {
26     return length * width;
27   }
28 }
```

Übungsblatt 03: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V5 Geometrische Formen II

Schreiben Sie eine Methode `double computeTotalArea` mit zwei Parametern `Circle[] circle` und `Rectangle[] rectangles`. Die Methode summiert die Flächeninhalte aller übergebenen geometrischen Formen in den beiden Arrays auf und gibt diese Summe zurück.

Lösungsvorschlag:

```
1  /**
2   * Computes the total area of all the specified geometry shapes.
3   *
4   * @param circles    an array of circles to add to the total area
5   * @param rectangles an array of rectangles to add to the total area
6   * @return the total area of the specified geometry shapes
7   */
8  double computeTotalArea(Circle[] circles, Rectangle[] rectangles) {
9    double sum = 0;
10
11   for (Circle circle : circles) { sum += circle.computeArea(); }
12   for (Rectangle rectangle : rectangles) { sum += rectangle.computeArea(); }
13   return sum;
14 }
```

Information: Alternativ zur foreach-Schleife, kann man eine gewöhnliche `for`-Schleife verwenden:

```
1  /**
2   * Computes the total area of the specified geometry shapes.
3   *
4   * @param circles    an array of circles to add to the total area
5   * @param rectangles an array of rectangles to add to the total area
6   * @return the total area of the specified geometry shapes
7   */
8  double computeTotalArea(Circle[] circles, Rectangle[] rectangles) {
9    double sum = 0;
10   for (int i = 0; i < circles.length; i++) {
11     // or sum = sum + circles[i].computeArea();
12     sum += circles[i].computeArea();
13   }
14   for (int i = 0; i < rectangles.length; i++) {
15     // or sum = sum + rectangles[i].computeArea();
16     sum += rectangles[i].computeArea();
17   }
18   return sum;
19 }
```

Übungsblatt 03: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V6 Spieglein, Spieglein ...

Wir nennen eine Gruppe von Elementen in einem Array Spiegel, wenn sie irgendwo im Array nochmal auftaucht, nur in umgekehrter Reihenfolge. Beispielsweise ist im Array `[7, 6, 5, 1, 9, 8, 5, 6, 7]` ein Spiegel vorhanden und zwar `[7,6,5]`. Schreiben Sie eine Methode `int maxMirror(int[] arr)`. Diese bekommt ein Array übergeben und gibt die Länge des größten Spiegels im übergebenen Array zurück. Gibt es keinen Spiegel so wird einfach `0` zurückgeliefert

> **Hinweis** Sie mit zwei Zeigern auf dem ersten und dem letzten Element. Vergleichen Sie nun paarweise die Elemente und überlegen Sie sich, wann Sie die beiden Zeiger weiter in die Mitte bewegen.

Lösungsvorschlag:

```
1  /**
2   * Returns the longest sequence in the array that has equal values at the positions i and
3   * array.length-i-1 in the array. If there is none, returns 0.
4   *
5   * @param arr array, where we try to find the longest sequence
6   * @return the longest sequence
7   */
8  int maxMirror(int[] arr) {
9    // Store the length of the maximum sequence
10   int max = 0;
11   // Store the length of the currently computed sequence
12   int current;
13
14   // Check the elements for a sequence by going forward in the array
15   for (int i = 0; i < arr.length; i++) {
16     // Check the elements for a sequence by going backward in the array
17     for (int j = arr.length - 1; j > i; j--) {
18       // Interval in the array that we want to check
19       int left = i;
20       int right = j;
21       current = 0;
22       // Compute the length of the sequence
23       while (left < right && arr[left] == arr[right]) {
24         current++;
25         left++;
26         right--;
27       }
28       // Check, if it is the longest subsequence and, if yes, overwrite the maximum
             sequence
29       if (max < current) {
30         max = current;
31       }
32     }
33   }
34   return max;
35 }
```

V7 Matrix-Multiplikation

Der folgende Code stellt beispielsweise die Matrix

$$\begin{pmatrix} 5 & 8 \\ 1 & -3 \end{pmatrix}$$

dar.

```
1 int[][] matrix = new int[2][2];
2 matrix[0][0] = 5;
3 matrix[0][1] = 8
4 matrix[1][0] = 1
5 matrix[1][1] = -3;
6
7 // alternativ und kuerzer: int[][] matrix = {{5,8},{1,-3}};
```

Sie sehen also, dass Sie einem Array in Java beliebig viele Dimensionen geben können.

Schreiben Sie eine Methode `int[][] matrixMul(int[][] mat1, int[][] mat2)`.

Die Methode bekommt zwei Matrizen, dargestellt durch zwei zwei-dimensionale Arrays, übergeben und gibt die resultierende Produktmatrix zurück. Sollte die Multiplikation aufgrund falscher Dimensionen nicht möglich sein, so geben Sie eine entsprechende Nachricht auf dem Bildschirm aus und liefern `null` zurück

Hinweis: Verwenden Sie drei ineinander geschachtelte `for`-Schleifen. Die erste iteriert über die Reihen von `mat1`, die zweite iteriert über die Spalten von `mat1` und die Reihen von `mat2` und die letzte iteriert über die Spalten von `mat2`.

Übungsblatt 03: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

```java
/**
 * Multiplies two matrices by doing matrix multiplication.
 *
 * @param mat1 the first matrix
 * @param mat2 the second matrix
 * @return the first matrix multiplied with the second matrix
 */
int[][] matrixMul(int[][] mat1, int[][] mat2) {
  int rowsMat1 = mat1.length;
  int colsMat1 = mat1[0].length;
  int rowsMat2 = mat2.length;
  int colsMat2 = mat2[0].length;
  // Resulting matrix
  int[][] mat3 = new int[rowsMat1][colsMat2];
  // Check, if the dimension is correct
  if (colsMat1 != rowsMat2) {
    // Error output print on console
    System.err.println("Wrong Dimensions");
    return null;
  }
  // Matrix multiplication
  for (int i = 0; i < rowsMat1; i++) {
    for (int j = 0; j < colsMat1; j++) {
      for (int k = 0; k < colsMat2; k++) {
        mat3[i][k] += mat1[i][j] * mat2[j][k];
      }
    }
  }
  return mat3;
}
```

Information: Ein zweidimensionales Array ist ein Array, dessen Komponententypen ebenfalls Arrays sind.

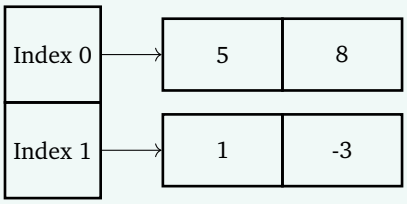

Abbildung 1: Visualisierung der Matrix als zweidimensionales Array

In der Abbildung 1 hat das Array zwei Elemente (am Index 0 und Index 1). Diese sind auch jeweils Arrays mit zwei Elementen. Beispielsweise würde `matrix[0]` ein Array mit zwei Elementen - und zwar `5` und `8` - zurückgeben. Wenn man die einzelnen Elemente innerhalb der inneren Arrays kriegen möchte, kann man bspw. folgenden Code verwenden:

- `matrix[0][0]` = 5
- `matrix[0][1]` = 8

Weitere Roboter-Klassen

In vielen Aufgaben reichen uns die eingeschränkten Methoden eines Roboters der `FopBot`-Werke nicht. Daher definieren wir uns neue Roboter, welche die technischen Anforderungen erfüllen. In den Foliensätzen zu `FopBot` haben Sie bereits Beispiele wie den `SymmTurner` Roboter dazu gesehen.

V8 CoinPutter

Implementieren Sie zunächst in der aus der Vorlesung bekannten Roboter-Klasse `SymmTurner` die `public void`-Methode `coinMove(int countOfSteps)`: Diese soll `countOfSteps` Schritte nach vorne gehen und vor jedem Schritt einen Coin ablegen. Sollte die geforderte Anzahl an Schritten größer sein als die Anzahl an Coins soll der Roboter einfach stehen bleiben und sich ausschalten. Verwenden Sie dazu die Ihnen aus der Vorlesung bekannte `public`-Methode `void turnOff()`. Sollten mehr Coins vorhanden sein als Schritte gefordert sind, soll er an seiner finalen Position alle verbleibenden Coins ablegen.

Übungsblatt 03: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

```java
/**
 * Moves the robot the specified number of steps and put a coin on each step.
 * If the robot does not have any coins, although there are steps remaining,
 * turn this robot off.
 *
 * @param numberOfSteps the number of steps this robot should take
 */
public void coinMove(int numberOfSteps) {
  while (hasAnyCoins() && numberOfSteps > 0) {
    move();
    putCoin();
    numberOfSteps--;
  }
  if (numberOfSteps > 0) {
    turnOff();
  }
  putALlCoins();
}

/**
 * Places all remaining coins of this robot on the ground.
 */
public void putAllCoins() {
  while (hasAnyCoins()) {
    putCoin();
  }
}
```

Übungsblatt 03: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V9 Richtungsdreher

In dieser Aufgabe soll eine neue Roboterklasse definiert werden, deren Roboter sich mittels eines einzigen Aufrufs in eine beliebige Richtung drehen können. Erstellen Sie dafür die Klasse `DirectionTurner`, die direkt von der Klasse `Robot` erbt und die parameter- und rückgabelosen `public`-Methoden `turnUp`, `turnRight`, `turnDown` und `turnLeft` so implementiert,dass der Roboter nach Aufruf einer dieser Methoden in die entsprechende Richtung blickt.

Übungsblatt 03: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

```java
/**
 * A direction turner robot that can turn to all available
 * {@link Direction}.
 */
public class DirectionTurner extends Robot {

  /**
   * Constructs a direction turner robot at the specified position
   * and direction. The robot contains
   * the specified amount of coins.
   *
   * @param x             the x-coordinate of the robot
   * @param y             the y-coordinate of the robot
   * @param direction     the direction of the robot
   * @param numberOfCoins the number of coins the robot have
   */
  public DirectionTurner(int x, int y, Direction direction, int numberOfCoins) {
    super(x, y, direction, numberOfCoins);
  }

  /**
   * Turns this robot to {@link Direction#UP}.
   */
  public void turnUp() {
    while (!isFacingUp()) { turnLeft(); }
  }

  /**
   * Turns this robot to {@link Direction#DOWN}.
   */
  public void turnDown() {
    while (!isFacingDown()) { turnLeft(); }
  }

  /**
   * Turns this robot to {@link Direction#LEFT}.
   */
  public void turnLeft() {
    while (!isFacingLeft()) { turnLeft(); }
  }

  /**
   * Turns this robot to {@link Direction#RIGHT}.
   */
  public void turnRight() {
    while (!isFacingRight()) { turnLeft(); }
  }
}
```

V10 TeamRobot

In dieser Aufgabe sollen Sie ihre erste eigene Roboterklasse von Grund auf implementieren. Erstellen Sie dazu eine neue Klasse `TeamRobot`, die die Klasse `Robot` erweitert, also von ihr erbt. Der Konstruktor der Klasse `TeamRobot` übernimmt die Parameter des Konstruktors der Oberklasse `Robot` und besitzt zusätzlich die Parameter `int left` und `int right`. Der Parameter `int left` gibt an, wie viele zusätzliche Roboter beim Aufruf des Konstruktors links neben des TeamRobots platziert werden. Der Parameter `int right` ist analog, für die Roboter rechts. Der `TeamRobot`, sowie die Roboter links und rechts von ihm bilden ein Team. Die zusätzlichen Roboter werden vom `TeamRobot` im Konstruktor erzeugt. Bekommt der `TeamRobot` einen Befehl, so soll dieser von allen Robotern im Team ausgeführt werden. Die zusätzlichen Roboter selbst sind dabei nicht ansprechbar, das heißt auf ihnen können keine Methoden aufgerufen werden. Überlegen Sie sich, wie Sie die Roboter des Teams in der `TeamRobot`-Klasse speichern können und wie Sie die Befehle die ein `TeamRobot` erhält, an alle Roboter im Team weiterreichen können. Die Befehle meinen hier die Methoden: `move()`, `turnLeft()`, `pickCoin()` und `putCoin()`.

> **Beispiel:** Beim Erstellen eines TeamRobots mit den Parametern `right = 1` und `left = 2` an der Position `(4, 4)`, werden zusätzlich 3 Roboter erstellt, die dem Team angehören, nämlich an der Position `(2, 4)`, `(3, 4)` (links) und `(5, 4)` (rechts).

Übungsblatt 03: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

```java
/**
 * A team robot class that contains multiple robots that form a team. All robots in the team
 * will execute the same operations.
 */
class TeamRobot extends Robot {

  /**
   * The robots to form a team.
   */
  private Robot[] partners;

  /**
   * Constructs a team robot at the specified position, with the specified direction and amount of coins.
   *
   * @param x             the x-coordinate of the center robot
   * @param y             the y-coordinate of the center robot
   * @param direction     the direction of the robots
   * @param numberOfCoins the amount of coins of the robots
   * @param left          the amount of robots to the left of the center robot
   * @param right         the amount of robots to the right of the center robot
   */
  public TeamRobot(int x, int y, Direction direction, int numberOfCoins, int left, int right) {
    super(x, y, direction, numberOfCoins);
    partners = new Robot[left + right];
    for (int i = 0; i < left; i++) {
      partners[i] = new Robot(x - i - 1, y, direction, numberOfCoins);
    }
    for (int i = 0; i < right; i++) {
      partners[i + left] = new Robot(x + i + 1, y, direction, numberOfCoins);
    }
  }

  @Override
  public void move() {
    super.move();
    for (Robot r : partners) {
      r.move();
    }
  }

  @Override
  public void turnLeft() {
    super.turnLeft();
    for (Robot r : partners) {
      r.turnLeft();
    }
```

```
48    }
49
50    @Override
51    public void putCoin() {
52      super.putCoin();
53      for (Robot r : partners) {
54        r.putCoin();
55      }
56    }
57
58    @Override
59    public void pickCoin() {
60      super.pickCoin();
61      for (Robot r : partners) {
62        r.pickCoin();
63      }
64    }
65  }
```

V11 The final Countdown ★★★

In Unix-basierten Systemen wird die Zeit traditionell als vorzeichenbehaftete 32 Bit Ganzzahl gespeichert, die seit dem 1. Januar 1970 vergangenen Sekunden repräsentiert.

Schauen Sie folgenden Java-Code an. Beheben Sie sämtliche eingebauten Fehler, um den Code lauffähig zu machen. Was müssen Sie im Code ändern, um die folgende Ausgabe zu erhalten?

"Am 19.1.2038 kommt es zu einem Ueberlauf des Unix Zeitstempels"

```java
public final class A {

  private int value1 = 0, value2 = 0;
  private final int value3 = 0;

  private int getValue1() {
    return value1;
  }

  private int getValue2() {
    return value2;
  }

  private void setValue1(int newValue1) {
    value1 = newValue1;
  }

  private void setValue2(int newValue2) {
    value2 = newValue2;
  }

  private final void changeValue3(final int newValue3) {
    value3 = 0;
  }
}

class B extends A {

  public void changeValue3(final int newValue3) {
    value3 = newValue3;
  }

  public static void main(String args[]) {
    B obj = new B();
    obj.setValue1(19);
    obj.setValue2(1);
    obj.value3 = 2038;

    String result = "Am " +
        getValue1() + "." +
        getValue2() + "." +
        obj.value3 +
        " kommt es zu einem Ueberlauf des Unix Zeitstempels.";

    System.out.println(result);
  }
}
```

Übungsblatt 03: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

```
1  public class A {
2
3    private int value1 = 0, value2 = 0;
4    public int value3 = 0;
5
6    public int getValue1() {
7      return value1;
8    }
9
10   public int getValue2() {
11     return value2;
12   }
13
14   public void setValue1(int newValue1) {
15     value1 = newValue1;
16   }
17
18   public void setValue2(int newValue2) {
19     value2 = newValue2;
20   }
21
22   public void changeValue3(final int newValue3) {
23     value3 = 0;
24   }
25 }
26
27 class B extends A {
28
29   public void changeValue3(final int newValue3) {
30     value3 = newValue3;
31   }
32
33   public static void main(String args[]) {
34     B obj = new B();
35     obj.setValue1(19);
36     obj.setValue2(1);
37     obj.value3 = 2038;
38
39     String result = "Am " +
40         obj.getValue1() + "." +
41         obj.getValue2() + "." +
42         obj.value3 +
43         " kommt es zu einem Ueberlauf des Unix Zeitstempels.";
44
45     System.out.println(result);
46   }
47 }
```

- Zeile 1: `public class A` sollte nicht final sein, da Klasse `B` ansonsten nicht von `A` erben kann.
- Zeile 4: `public int value3`, soll nicht `final` sein und `public`, `protected` (in Subklassen und im Package sichtbar) oder leer sein (standardmäßig im Package sichtbar), da wir ansonsten mit der Methode `changeValue3` den Wert nicht verändern können und auch in der Klasse `B` darauf nicht zugreifen können.
- Zeile 6, 10, 14, 18, 22: Der Zugriffmodifikator muss entweder public oder `protected` (in Subklassen und im Package sichtbar) oder leer sein (standardmäßig im Package sichtbar), da Klasse `B` sonst nicht auf diese Methoden zugreifen kann.
- Zeile 22: `public void` (ohne `final`), da wir ansonsten die Methode in der Klasse `B` nicht überschreiben können.
- Zeile 40, 41: `obj.getValue1()` und `obj.getValue2()`, da die Objektmethoden nur über eine Instanz der Klasse `A` oder `B` angesprochen werden können.

Mehr Informationen zu Zugriffmodifikatoren können Sie unter folgendem Verweis finden:

`https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html`