Name: *Matrikelnummer:*

Programmierung

Klausur 1 (Wintersemester 2018/19)

12. März 2019

Answer: *With Solutions*

Hinweise (*English translation below*)

- **Bitte schlagen Sie die Klausur erst auf, sobald Sie dazu aufgefordert werden.**
- Einlesezeit: 10 Minuten. Fragen zur Klausur sollten innerhalb dieser Zeit gestellt werden. Wenn Sie eine Frage haben, melden Sie sich, und es wird jemand zu Ihnen kommen. **Anschließende** Bearbeitungszeit: 90 Minuten
- Mögliche Gesamtpunktzahl: 100 Punkte (+ 10 Bonuspunkte).
- Überzeugen Sie sich davon, dass Ihre Klausur vollständig ist.
- Legen Sie Ihren Studierendenausweis sowie einen Lichtbildausweis vor sich auf den Tisch. Während der Klausur werden wir Ihre Identität kontrollieren.
- Es sind keinerlei Hilfsmittel (Taschenrechner, Skripte, Bücher, Notizen, Telefone, etc.) für diese Klausur erlaubt. Falls Sie Papier brauchen, geben Sie uns Bescheid, statt eigenes zu benutzen. Bitte schalten Sie Ihre Telefone ab und verstauen Sie Smart Watches in Ihrem Rucksack. Wenn Sie gegen diese Regeln verstoßen, riskieren Sie, von der Prüfung ausgeschlossen zu werden und durchzufallen.
- Schreiben Sie Ihre Antworten auf die Aufgabenzettel. Sie können Antworten gegebenenfalls auf der letzten leeren Seite fortsetzen, bitte weisen Sie dann entsprechend darauf hin. Sollte der Platz immer noch nicht ausreichen, fragen Sie uns nach zusätzlichem Papier.
- Sie dürfen die Fragen auf deutsch oder englisch beantworten.
- Bitte schreiben Sie auf **jedes Blatt**, das Sie abgeben, Ihren Namen und (falls vorhanden) Ihre Matrikelnummer.
- Bis 20 Minuten vor Ende der Klausur dürfen Sie vorzeitig abgeben. Wenn Sie nicht vorzeitig abgeben, warten Sie bitte an Ihrem Platz bis Ihre Klausur zusammengeheftet ist und eingesammelt wird.
- Die korrigierten Klausuren können am 18.03.2019 zwischen 14 und 17 Uhr in CAP 4, Raum 1001, eingesehen werden. Die exakte Zeit für Sie hängt von Ihrer Rechnerübung ab. Bitte schauen Sie dafür ins Wiki.

English translation of instructions above—such translations are also provided for the exam problems

- **Please do not turn the page before you are told to do so.**
- Reading time: 10 minutes. Questions concerning the problems should be asked during that time. If you have a question, raise your hand, and somebody will come to you to listen to your question. **Afterwards** time for problem solving: 90 minutes.
- The total maximum score: 100 points (+ 10 bonus points).
- Ensure that your copy of the exam is complete.
- Put a photo ID card and your student ID card in front of you. We will examine it during the exam.
- You are not allowed to use anything other than a pen to write with. In particular, you are not allowed to use the book, any printouts, or electronic devices (which includes phones and smart watches). Do not use your own paper to write on; instead, ask us for paper, we have plenty. Breaking these rules means risking a failing grade.
- Use the space provided in the assignments to write down your answers. You may continue your answers on the very last, empty page; please make a note if you do so. If you still run out of space, ask us for additional paper.
- You may answer in English or German.
- Please put your name and immatriculation number (*Matrikelnummer*), if you have one, onto **every sheet** that you hand in.
- You are allowed to hand in early until 20 minutes before the end of the exam. If you do not hand in early, please wait at your seat until your exam has been collated and collected.
- You can examine your corrected exam on March 18, 2019 between 2pm and 5pm in CAP 4, room 1001. The exact time you should show up at depends on which practical class you were assigned to during the semester. Our wiki contains further details.

Aufgabe 1 (12 Punkte) Kreuzen Sie bei den folgenden Fragen bitte alle zutreffenden Optionen an. Für jede richtig beantwortete Frage, bei der also genau die zutreffenden Optionen angekreuzt sind, gibt es einen Punkt. Die Zahl der jeweils zutreffenden Optionen kann zwischen "keine" und "alle" variieren; das heißt, zufälliges ankreuzen lohnt sich nicht.

Tick all correct options in the following questions. For each correctly answered question, where exactly the correct options are ticked, you earn one point. The number of correct options may vary arbitrarily between "none" and "all." It thus does not pay to randomly tick options.

1. Welche der folgenden Codeausschnitte sind valide Java-Anweisungen?

   Which of the following statements are valid Java code?

   ☐ `int pi = 3.14;` ☒ `double pi = 3;` ☒ `char pi = 3;` ☐ `String pi = 3;`

2. Welche der folgenden Klassen erweitern, direkt oder indirekt, die Klasse `Object`?

   Which of the following classes extend the `Object` class, either directly or indirectly?

   ☒ `ConsoleProgram` ☒ `GObject` ☒ `RandomGenerator` ☒ `Integer`

3. Welche der folgenden Arten von Variablen werden auf dem *Heap* abgelegt?

   Which of the following kinds of variables are saved on the *heap*?

   ☒ Instanzvariablen
   Instance variables

   ☐ Klassenvariablen
   Class variables

   ☐ Statische Variablen
   Static variables

   ☐ Lokale Variablen
   Local variables

4. Was ist das Resultat von `7 & 9`?

   What is the result of `7 & 9`?

   ☒ 1 ☐ 2 ☐ 3 ☐ 5

5. Welche der folgenden Aussagen über Klassen sind korrekt?

   Which of the following statements about classes are correct?

   ☐ Klassen können beliebig viele direkte Superklassen haben.
   Classes can have arbitrarily many direct superclasses.

   ☒ Klassen können beliebig viele Interfaces implementieren.
   Classes can implement arbitrarily many interfaces.

   ☐ Klassen müssen immer einen explizit hingeschriebenen Konstruktor haben.
   Classes must always have an explicitly defined constructor.

   ☐ Klassen haben immer eine Superklasse.
   Classes always have a superclass.

6. Konstanten sind immer...

   Constants are always...

   ☐ ...**`static`** ☐ ...Instanzen von Klassen
   ...instances of classes

   ☒ ...**`final`** ☐ ...außerhalb der deklarierenden Klasse sichtbar
   ...visible outside the declaring class

7. Gegeben sei folgende Grammatik als EBNF. Welche der unten aufgeführten Worte sind Teil der durch die Grammatik definierten Sprache?

   Given the following grammar in EBNF, which of the words below are part of the language the grammar defines?

```
Or ::= Or '|' And | And
And ::= And '&' Par | Par
Par ::= '(' And ')' | Dig
Dig ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

   ☒ `1 | 2 & 3` ☒ `1 & 2 | 3` ☐ `1 & (2 | 3)` ☒ `(1 & 2) | 3`

8. Welche der folgenden Aussagen über `null` treffen zu?

   Which of the following statements about `null` are correct?

   ☐ Es ist sinnvoll, per `obj.equals(null)` zu prüfen, ob `obj` gleich `null` ist.
   It makes sense to check whether `obj` is `null` by writing `obj.equals(null)`.

   ☐ Variablen primitiver Typen können den Wert `null` annehmen.
   `null` can be assigned to variables of primitive type.

   ☒ Variablen mit Objekttypen können den Wert `null` annehmen.
   `null` can be assigned to variables of object type.

   ☒ Nicht explizit initialisierte Instanzvariablen mit Objekttyp bekommen von Java den Initialwert `null`.
   Instance variables of object type that are not explicitely initialized are initially `null`.

Aufgabe 2 (12 Punkte)

1. Was ist ein *Term*?

   What is a *term*? ***Answer:*** *Used in expression. Can be constant, variable name, method call, or expression in parentheses.*

2. Was ist eine *Variablendeklaration*?

   What is a *variable declaration*? ***Answer:*** *Declaration establishes name and type of variable, possibly also initial value.*

3. Was ist *Assoziativität*?

   What is *associativity*? ***Answer:*** *Associativity determines the order of evaluation among operators with the same precedence.*

4. Was ist ein *Scope*?

   What is a *scope*? ***Answer:*** *The scope of a variable is where it exists. Local variables exist from declaration until the end of the enclosing block/method; variables declared in loop header exist in the loop body.*

5. Warum ist eine `for`-Schleife nicht die beste Art von Schleife, um so lange zu iterieren, bis ein vom Benutzer wiederholt eingegebener Wert valide ist?

   Why is a `for` loop not the best kind of loop to repeatedly ask the user for a value until that value is considered valid? ***Answer:*** *Because we don't know how many attempts the user will need.*

6. Was ist eine *Prädikaten-Methode*?

   What is a *predicate method*? ***Answer:*** *A method that returns a boolean value.*

7. Was sind *lokale Variablen*?

   What are *local variables*? ***Answer:*** *Variables declared in a method.*

8. Was ist ein *Paket* in Java?

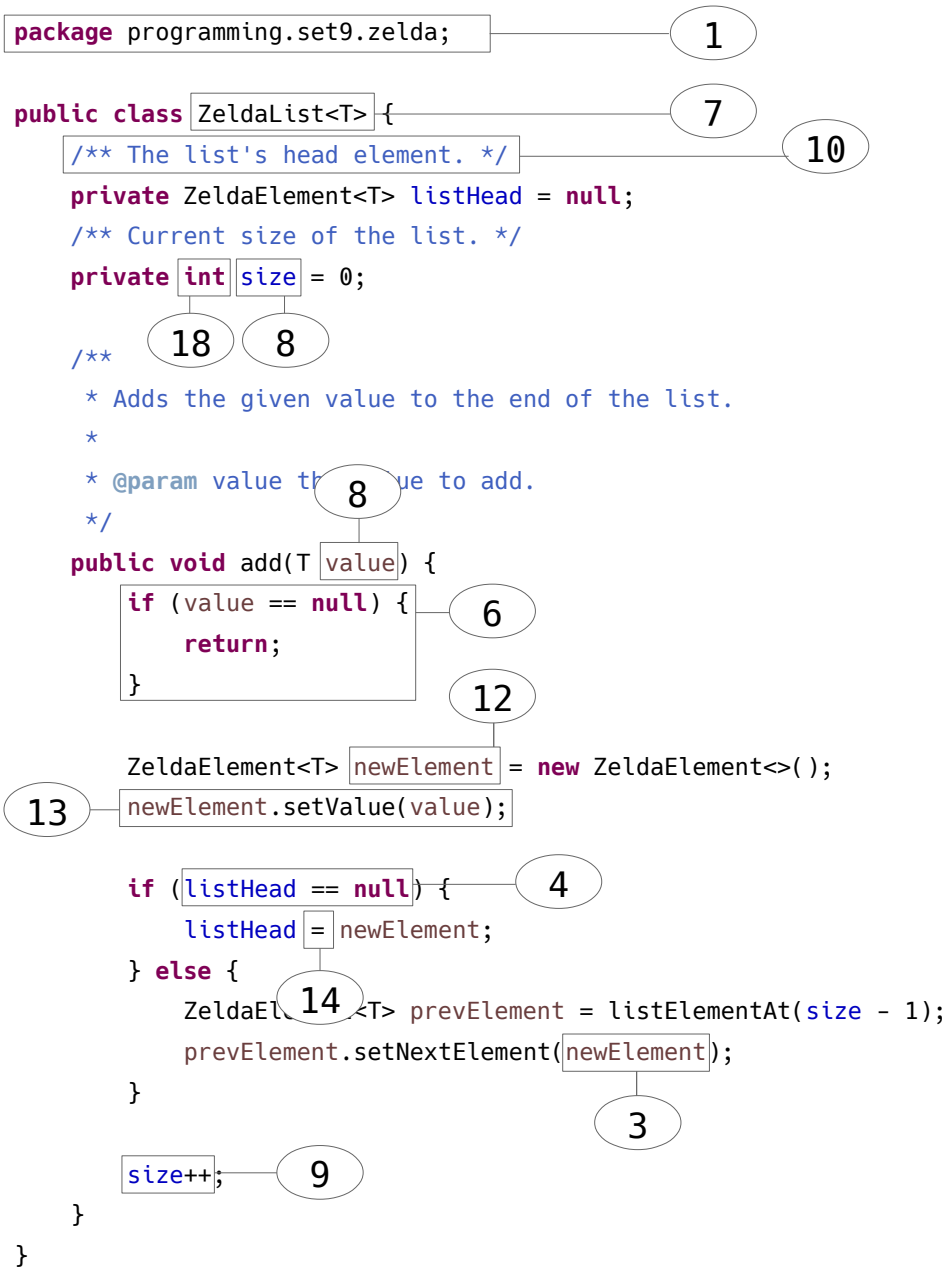
   What is a *package* in Java? ***Answer:*** *Collection of related classes that have been released as a coherent unit.*

Aufgabe 3 (18 Punkte) Im Folgenden sehen Sie ein Stück Java-Code. Schreiben Sie in jedes der vorgesehenen Felder die Zahlen der dazu passenden Begriffe aus der Liste. Pro Feld können mehrere Zahlen zutreffen. Es müssen nicht alle Begriffe Verwendung finden.

Consider the following piece of Java code. Annotate each of the highlighted parts of code with the corresponding term or terms from the list of terms below by writing the appropriate number(s) into the respective circle. There may be terms that will go unused.

1. Package declaration
2. Access modifier
3. Argument
4. Boolean expression
5. Class variable
6. Conditional statement
7. Generic class
8. Instance variable
9. Integer expression
10. Javadoc comment
11. Literal
12. Local variable
13. Method call
14. Operator
15. Parameter
16. Return statement
17. Return type
18. Type

Answer:



Aufgabe 4 (40 Punkte) Sie haben sich zum Ziel gesetzt, Labyrinthe zu bauen, und wollen dafür eine Klasse schreiben, welche Ihnen dabei hilft. Um sie zu benutzen muss man sie mit der gewünschten Größe des Labyrinths instanziieren, so viele Gänge hinzufügen wie man möchte, und sie dann darum bitten, einen String zu generieren, welcher das Labyrinth darstellt (inklusive einer Umrandung). Der unten stehende Beispielcode soll in der daneben abgebildeten Konsolenausgabe resultieren. Das Hinzufügen von Gängen, die nicht komplett innerhalb des Labyrinths liegen, soll beim Aufruf von `addCorridor(...)` eine `IllegalArgumentException` erzeugen.

Schreiben Sie die Klasse `LabyrinthGenerator` entsprechend dieser Spezifikationen. Um die volle Punktzahl zu erreichen denken Sie daran, Ihren Code zu kommentieren. Sie brauchen benutzte Klassen nicht importieren. Sie können annehmen, dass `Direction` eine Enumeration ist, welche genau die Konstanten `DOWN`, `UP`, `LEFT` und `RIGHT` enthält.

You decided to build labyrinths and want to write a class that supports you in your task. To use that class, one has to instantiate it with the labyrinth's desired size, add as many corridors as desired, and ask it to generate a string which depicts the labyrinth (including a border). The sample code below should produce the console output printed next to it. Trying to add a corridor which is not completely contained in the labyrinth should cause the `addCorridor(...)` method to throw an `IllegalArgumentException`.

Write the class `LabyrinthGenerator` according to these specifications. To receive full credit, remember to comment your code. You do not need to import classes you use. You can assume that `Direction` is an enumeration which contains exactly the enumeration constants `DOWN`, `UP`, `LEFT`, and `RIGHT`.

```
LabyrinthGenerator lg = new LabyrinthGenerator(8, 4);
lg.addCorridor(0, 0, Direction.RIGHT, 5);
lg.addCorridor(7, 2, Direction.LEFT, 4);
lg.addCorridor(3, 0, Direction.DOWN, 4);
lg.addCorridor(6, 1, Direction.UP, 2);
System.out.println(lg.toString());
```

```
##########
#     # ##
#### ## ##
####     #
#### #####
##########
```

Answer:

```
/**
 * Generates labyrinths in ASCII art.
 */
public class LabyrinthGenerator {
    /** Height of our labyrinth. */
    private final int height;
    /** Width of our labyrinth. */
    private final int width;
    /** Contains {@code true} where the labyrinth is walkable. */
    private final boolean[][] labyrinth;

    /**
     * Creates a new generator for a labyrinth of the given size.
     *
     * @param width the labyrinth's width.
     * @param height the labyrinth's height.
     */
    public LabyrinthGenerator(int width, int height) {
        this.height = height;
        this.width = width;
        labyrinth = new boolean[height][width];
    }

    /**
     * Adds a corridor to the labyrinth.
     *
     * @param x x coordinate of the corridor's start point.
     * @param y y coordinate of the corridor's start point.
     * @param dir direction where the corridor is heading.
     * @param length the corridor's length (number of fields).
     */
```

```
    public void addCorridor(int x, int y, Direction dir, int length) {
        // Check coordinates
        if (x < 0 || x >= width || y < 0 || y >= height) {
            throw new IllegalArgumentException();
        }

        // Determine where to go in each iteration
        int deltaX = 0;
        int deltaY = 0;

        switch (dir) {
        case UP:
            deltaY = -1;
            break;
        case DOWN:
            deltaY = 1;
            break;
        case LEFT:
            deltaX = -1;
            break;
        case RIGHT:
            deltaX = 1;
            break;
        }

        // Determine if the corridor is contained in the labyrinth
        int endX = x + deltaX * (length - 1);
        int endY = y + deltaY * (length - 1);

        if (endX < 0 || endX >= width || endY < 0 || endY >= height) {
            throw new IllegalArgumentException();
        }

        // Add the corridor
        for (int i = 0; i < length; i++) {
            labyrinth[y][x] = true;
            x += deltaX;
            y += deltaY;
        }
    }

    @Override
    public String toString() {
        String result = "";

        // Top border
        for (int i = 0; i < width + 2; i++) {
            result += "#";
        }
        result += "\n";

        // Each row
        for (int row = 0; row < height; row++) {
            result += "#";

            for (int col = 0; col < width; col++) {
                if (labyrinth[row][col]) {
                    result += " ";
```

```
                } else {
                    result += "#";
                }
            }

            result += "#\n";
        }

        // Bottom border
        for (int i = 0; i < width + 2; i++) {
            result += "#";
        }

        return result;
    }
}
```

Aufgabe 5 (18 Punkte) Bestimmen Sie für jedes der nachfolgenden Code-Beispiele, ob es eine Eingabe für `x` so gibt, dass der Code genau ein Mal „success" und nichts anderes ausgibt und dabei keinen Fehler verursacht. Wenn es mehrere gültige Eingaben für `x` gibt, reicht es, eine anzugeben. Wenn es keine Eingabe gibt, streichen Sie den Lösungsbereich erkennbar durch.

For each of the following code snippets, determine whether there are any values of `x` that can be entered so that the code will print "success" exactly once, without causing any errors and without printing anything else. If there are multiple values of `x` that will work, just give one of them. If there are no values of `x` that will work, strike through the solution area.

```
int x = readInt();
if (x > 7 || x / 0 == x) {
  println("success");
}
```

Answer: *$x > 7$*

```
int x = readInt();
if (1 - 3 - 5 - x == -10) {
  println("success");
}
```

Answer: *3*

```
int x = readInt();
int y = x % 4;
for (int i = 1; i < y; i++) {
  x *= i + 1;
}

if (x % 2 == 0) {
  println("success");
}
```

Answer: *All numbers for which x mod 4 is not 1.*

```
int x = readInt();
if ((x + 1) % 2 == 0) {
  println("success");
}
if ((x - 1) % 2 == 0) {
  println("failure");
}
```

Answer: *No solution.*

```
int x = readInt();
while (x >= 0) {
  x = x % 10;

  if (x == 0) {
    println("success");
  }
}
```

Answer: *No solution.*

```
int x = readInt();
int a = 1, b = 1, c = 1;
while (c <= x) {
  c = a + b;
  a = b;
  b = c;
  if (x == b) {
    println("success");
  }
}
```

Answer: *Any Fibonacci number starting at 2.*

Aufgabe 6 (10 Bonus Punkte) Gegeben sei das folgende Programm. Geben Sie an, was auf der Konsole ausgegeben wird wenn `args` bei Aufruf der `main(...)`-Methode die vier Strings `"The"`, `"Dark"`, `"Knight"` und `"Rises"` in genau dieser Reihenfolge enthält. Geben Sie weiterhin den Inhalt von `map` bei Aufruf der `toString()`-Methode an.

Consider the following program. Write down what will be printed out on the console if `args` is an array consisting of the four Strings `"The"`, `"Dark"`, `"Knight"` and `"Rises"` (in that order). Also write down the content of `map` when `toString()` is called.

```
import java.util.HashMap;
import java.util.Map;

public class Mystery {
    private String str;
    private static Map<String, Integer> map = new HashMap<>();
    private static int integer;

    private Mystery(String s) {
        str = s;
        integer = str.length();
        map.put(str, integer);
    }

    private Mystery combine(Mystery other) {
        map.remove(other.str);
        integer += other.integer;
        return this;
    }

    @Override
    public String toString() {
        return integer + " " + map.size();
    }

    public static void main(String[] args) {
        Mystery[] mysteries = new Mystery[args.length];
        for (int i = 0; i < mysteries.length; i++) {
            mysteries[i] = new Mystery(args[i]);
        }

        Mystery result = mysteries[0];
        for (int i = 1; i < mysteries.length - 1; i++) {
            result = result.combine(mysteries[i]);
        }
        result = new Mystery(args[mysteries.length - 1]);

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

Answer: *Output:* `5 2`
Content of `map`*:*
`"The" -> 3, "Rises" -> 5`