Auftrag AB9: Methoden mit Parametern

Ein Platz für ein neues Endlager ist gefunden! In einem Bergwerk soll dies geschaffen werden. Sprengarbeiten müssen vorgenommen und Brennstäbe eingelagert werden. Eine Aufgabe wie gemacht für unsere Roboter … Eine Einsatzleiterin wurde bestimmt, die anstehenden heiklen Arbeiten zu koordinieren.

Ziel: Du kannst Probleme mit Hilfe von flexibel einsetzbaren Methoden lösen. Diese Methoden enthalten Parameter, mit denen sich das Verhalten der Roboter flexibel steuern lässt.

Bei manchen Einsätzen müssen Roboter situationsgerecht agieren können. So legt beispielsweise ein Roboter über den Methodenaufruf `ablegen(…)` eine Schraube, ein anderer einen Brennstab ab. Welchen Gegenstand der einzelne Roboter ablegen soll, bekommt er als sogenannten **Parameter** innerhalb der **Parameterklammer** `(  )` mitgeteilt. So bewirkt `ablegen("Schraube")` etwas anderes als `ablegen("Brennstab")`. Das kennst du schon seit AB3 Aufgabe 1.

Als Parameter wird der Methode `ablegen(…)` ein `String`, also ein Text mitgegeben. Dieser Text muss, wie du schon gelernt hast, in Java immer in Anführungszeichen geschrieben werden.

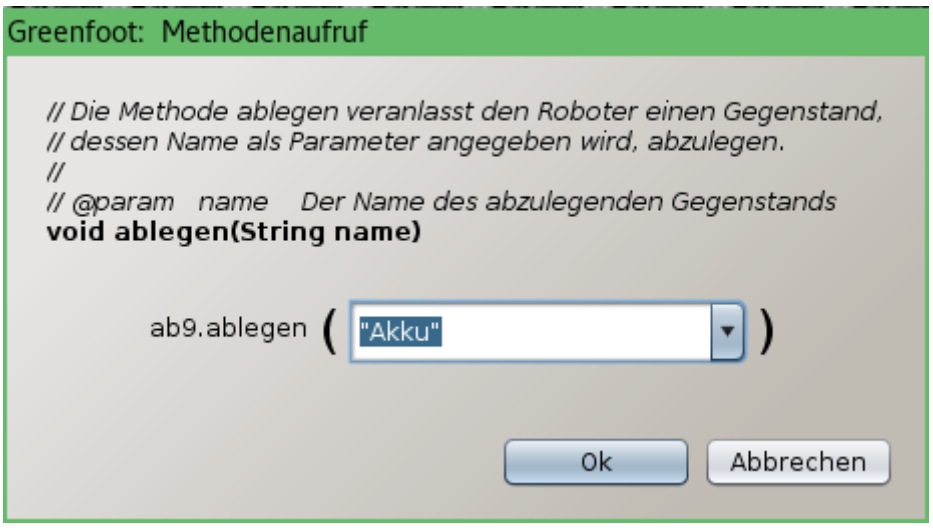
Als weitere Methoden mit Parametern hast du beispielsweise schon diese aufgerufen:

- `benutze("Feuerloescher")`
- `istVorne("Fass")`
- `istAufGegenstand("Brennstab")`

Um den Robotern diese Variabilität beizubringen, muss der Kopf der Methodenbeschreibung (oder auch *Signatur* der Methode genannt) wie folgt lauten:

```
public void ablegen(String name) { ... }
```

Das Schlüsselwort `String` in der Parameterklammer hinter dem Methodennamen besagt, dass bei Befehlserteilung ein Text anzugeben ist, damit der Befehl ausgeführt werden kann.



Es wird also der Name des Gegenstandes erwartet, der abgelegt werden soll. Wenn du keinen Text eingibst, weiß der Roboter nicht, welchen Gegenstand er ablegen soll. Es erscheint eine

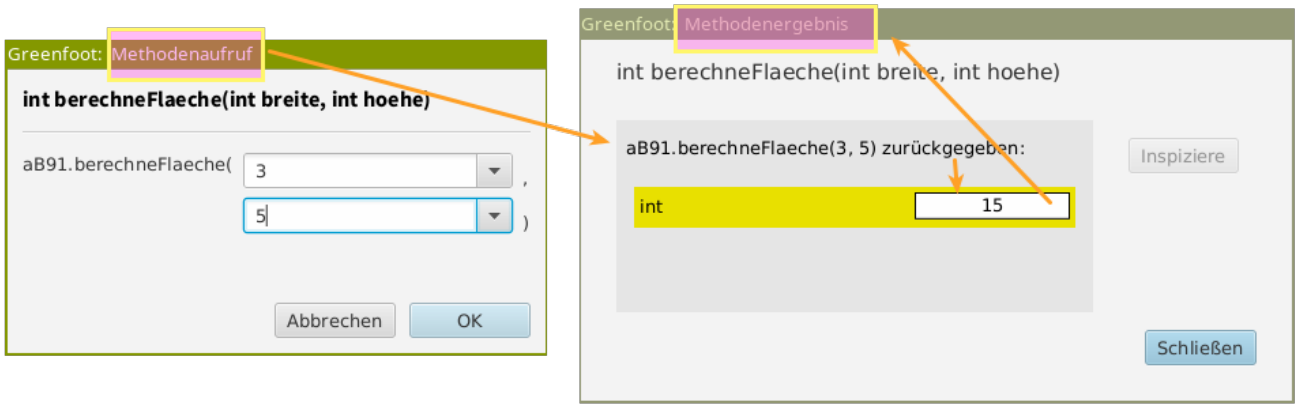
Fehlermeldung. Eine Fehlermeldung erscheint ebenfalls, wenn der Eingabewert nicht vom Typ String, also z.B. nicht in Anführungszeichen geschrieben ist.

Als Parameter sind alle Variablentypen denkbar. So erwartet beispielsweise die Methode mit der Signatur `public void macheWasXMal(int anzahl)` einen Integer, also eine Ganzzahl. Bei der Befehlserteilung ist also eine Zahl anzugeben, damit der Befehl ausgeführt werden kann. Gültige Methodenaufrufe wären beispielsweise `macheWasXMal(7)` oder `macheWasXMal(1234)`. Sogar `macheWasXMal(-3)` wäre erlaubt, nach einem Sinn müssten wir jedoch suchen 😉.

Manchmal ist es auch erforderlich mehrere Parameter anzugeben. So hat die Methode

```
int berechneFlaeche(int breite, int hoehe)
```

zwei Parameter vom Typ int, die zur Flächenberechnung genutzt werden. Der Methodenaufruf `berechneFlaeche(3, 5)` würde somit den Wert 15 als Methodenergebnis liefern (siehe Bilder).



Beim Methodenaufruf der Methode `melde(String text, boolean istWichtig)` wird an erster Position die Eingabe eines Textes erwartet, an zweiter Position die Eingabe eines Wahrheitswertes true oder false, je nachdem, ob die Meldung besonders wichtig ist oder nicht.

Aufgaben:

Aufgabe 1

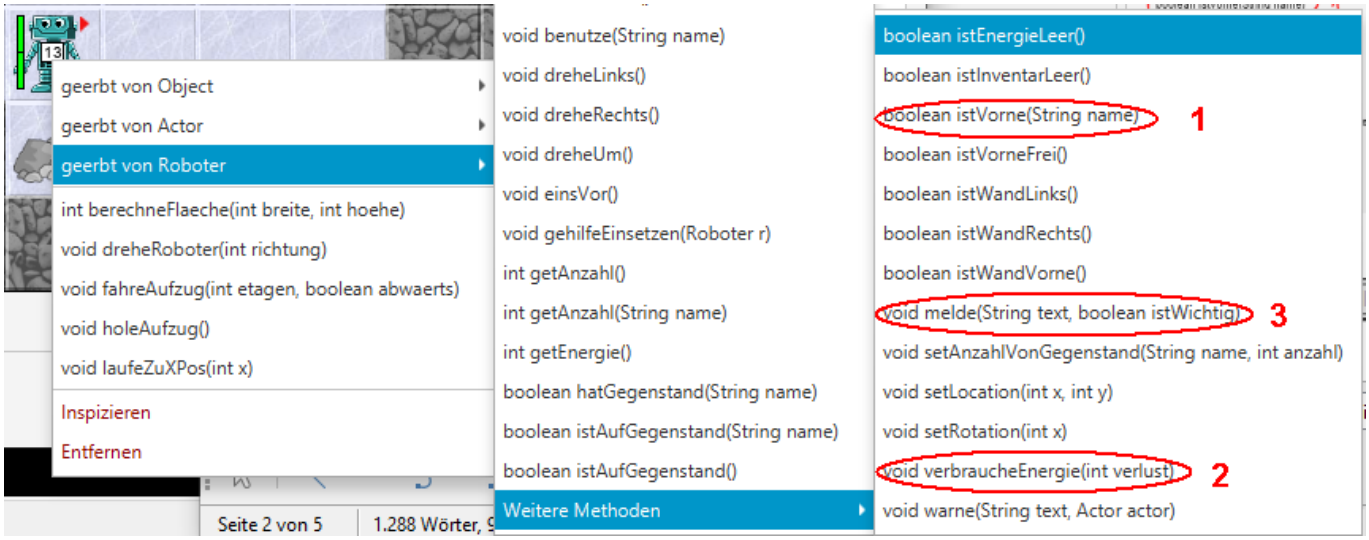
- Markiere im untenstehenden Bild alle Methoden, die beim Aufruf eine Eingabe verlangen.
- Welche Methode hat die längste Parameterliste?
- Woran erkennst du, ob eine Methode einen Wert zurück gibt?
- Welche zwei Methoden haben sowohl Parameter, als auch einen Rückgabewert?

```
void ablegen(String name)
int berechneFlaeche(int breite, int hoehe)
float berechneVolumen(float breite, float hoehe, float tiefe)
String gibAntwortsatz()
int gibAnzahlBrennstaebe()
float gibRadioaktivitaet()
void holeBombeUndKehreZurueck(int xPos, int yPos)
boolean istDunkel()
void legeAnzahlBrennstaebe(int anzahl)
void legeBrennstab(int x, int y, boolean aufKontaktplatte)
void legeBrennstaebe(int pos1X, int pos1Y, int pos2X, int pos2Y, int pos3X, int pos3Y)
void sprenge(int startX, int startY)
```

Aufgabe 2

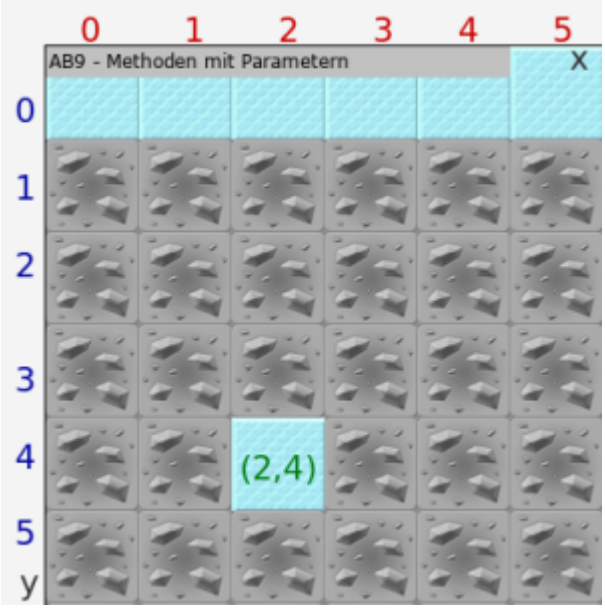
Du bist jetzt in Greenfoot im Level „AB9 – Methoden mit Parametern“. Alle Roboter haben (in der Klasse Roboter) schon einige Methoden mit Parametern implementiert.

Teste an einem AB9 die drei unten markierten Methoden (indem du bei einem Roboter rechts klickst und auf geerbt von Roboter → Weitere Methoden gehst).



Beschreibe jeweils kurz, welche Aufgabe die jeweilige Methode erfüllt.

Im Bergwerk bringt ein Aufzug die Roboter in das richtige Stockwerk. Durch einen „Schalter“ kann der Aufzug in das oberste Stockwerk gerufen werden. Dafür gibt es schon die Methode `holeAufzug()`.



Aufgabe 3

Drehe Roboter: Vervollständige die Methode

```
public void dreheRoboter(int richtung) { ... }
```

die den Roboter in die angegebene Richtung dreht (0=Blick nach rechts, 90=Blick nach unten, ...). Mit `getRotation()` kann man die aktuelle Richtung erfragen.

Tipp: Drehe den Roboter so lange, bis `richtung` erreicht ist.

Wie du sicherlich schon erkannt hast, schauen wir nun nicht mehr auf die Roboterwelt von oben, sondern wir betrachten einen Querschnitt eines Bergwerks. Hier spielt – wie im wirklichen Leben – die **Schwerkraft** eine wichtige Rolle. Pass auf, dass deine Roboter nicht in den Aufzugschacht fallen.

Aufgabe 4

Laufe zu: Vervollständige die Methode

```
public void laufeZuXPos(int x)
```

die den Roboter zu der angegebenen x-Koordinate laufen lässt. Die y-Position kann sich auch aufgrund der Schwerkraft verändern.

Tipp: Durch Rechtsklick auf die Welt (blauer Hintergrund) kannst du die Methode `zeigeKoordinaten(` aufrufen, somit erkennst du schnell, wie die Koordinaten eines Feldes lauten.

Der alte Grubenaufzug funktioniert etwas eigentümlich und muss noch programmiert werden. Bisher muss ein Roboter nach unten bzw. nach oben schauen und einsVor gehen, damit der Aufzug ein Stockwerk nach unten bzw. nach oben fährt.

Aufgabe 5

Fahre Aufzug: Vervollständige die Methode

```
public void fahreAufzug(int stockwerke, boolean abwaerts)
```

die einen Roboter, der auf einem Aufzug steht (`istAufGegenstand("Aufzug")`) die angegebene Anzahl von Stockwerken (ein Stockwerk entspricht einem Schritt) abwärts oder aufwärts fahren lässt. Diese Methode soll funktionieren, egal in welche Richtung der Roboter am Anfang schaut. Steht der Roboter nicht auf einem Aufzug, soll nichts passieren.

Bedenke: Die Roboter können in leere Aufzugsschächte stürzen, da man in diesem Level nicht von oben sondern von vorne auf die Welt schaut.

Aufgabe 6

Implementiere die Methode

```
public void fahreInsStockwerk(int stockwerk)
```

Dabei werden die Stockwerke vom Boden ab abwärts gezählt (blaue Koordinate).

Aufgabe 7

Bombe sichern: Implementiere die Methode

```
public void holeBombeUndKehreZurueck (int xBombe, int yBombe)
```

die einen Roboter, der auf dem Aufzug steht, zur angegeben Position laufen lässt und dort eine Bombe einsammelt. Danach soll er zum Aufzug zurück kehren.

Aufgabe 8

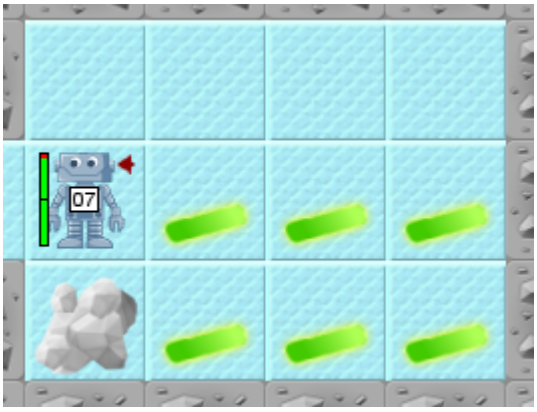
Nun soll ein AB9-Roboter einen Bereich im Bergwerk sprengen. Implementiere dazu die Methode

```
public void sprenge (int xPos, int yPos)
```

die einen Roboter, der auf dem Aufzug steht, zur übergebenen (x|y)-Position laufen lässt, mit `benutze("Bombe")` die Bombe zündet.

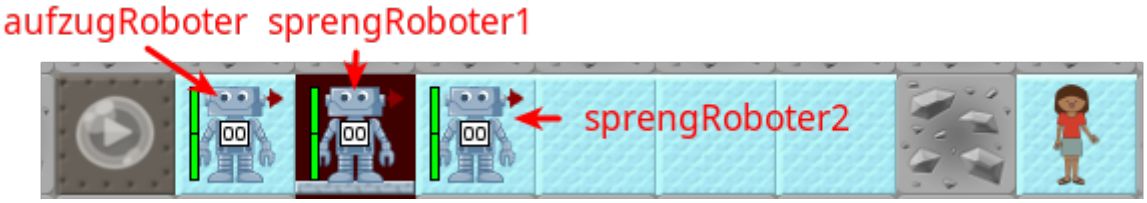
Aufgabe 9

Legen: Der Roboter soll einen Raum mit einer Zweierreihe von Brennstäben (von denen er ausreichend dabei hat) belegen. Implementiere dazu eine Methode, die als Parameter die Breite des Raumes bekommt.



Die Schwerkraft bewirkt, dass ein abgelegter Brennstab nach unten rutscht. D.h. legt man einen Brennstab auf einen Felsen, rutscht dieser runter, wenn kein Fels oder anderer Gegenstand da liegt. Beachte, dass der Roboter sich nicht mehr nach oben bewegen kann.

Nun kommt der **Einsatzleiter** ins Spiel. Das ist das kleine Männchen oben rechts. Der Einsatzleiter kennt die vier Roboter. Der erste ganz links heißt `aufzugRoboter`, der rechts daneben `sprengRoboter1`, dann `sprengRoboter2` und unten steht der `legeRoboter`.



Im Folgenden muss der Einsatzleiter jeweils Methoden bei einem der vier Roboter aufrufen. Im Quelltext schreibt man dazu beispielsweise folgenden Befehl:

```
sprengRoboter2.fahreAufzug(2,true);
```

Man schreibt also den Namen des Roboters, gefolgt von einem Punkt. Danach kommt der Name der Methode und die dazugehörigen Parameter. (Merkhilfe für die Reihenfolge: Wer macht was?)

Drückt man die Tastenkombination STRG+Leertaste nach der Eingabe des Punktes, so bekommt man eine Liste aller Methoden, die man bei diesem Roboter aufrufen kann.

Aufgabe 10

Öffne die Klasse `EinsatzLeiter` und vervollständige die Methode

```
public void holeBombeUndSprenge (int xBombe, int yBombe, int xPos, int yPos)
```

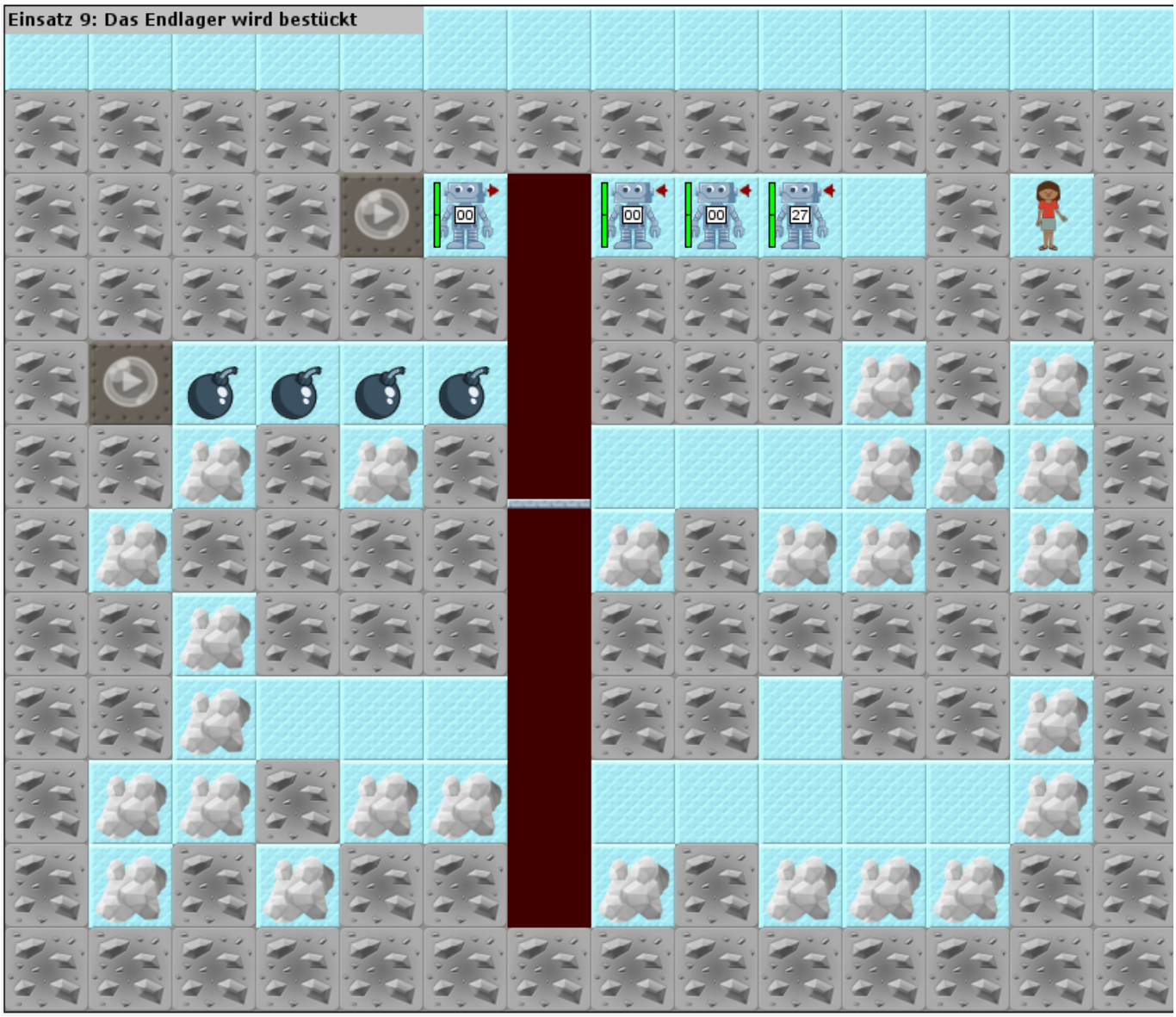
Dabei sind `(xBombe|yBombe)` die Koordinaten, wo sich eine Bombe befindet und `(xPos|yPos)` die Koordinaten der Sprengposition. Der Einsatzleiter soll dazu dem `aufzugRoboter` und dem `sprengRoboter1` die passenden Befehle geben! (Wie das geht, siehst du im Quelltextbeispiel).

Einsatz 9

Im Folgenden soll deine Einsatzleiterin den Einsatz im Endlager koordinieren. Die AB9-Roboter haben alles gelernt, was sie für diesen Einsatz benötigen:

Sie können in einem Stollen Sprengarbeiten vornehmen und einen freigewordenen Bereich mit Brennstäben belegen. Aber Achtung, die Ressourcen sind knapp – sowohl die Anzahl der Sprengsätze, als auch die Energie der Roboter. Die Arbeit unter Tage ist nicht nur für Menschen anstrengend...

Deine Aufgabe ist es nun, den Einsatzleiter so zu programmieren, dass sie den Robotern klare Anweisungen gibt, um die drei verschütteten Stollen zu erweitern. Der `legeRoboter` soll dann seine Brennstäbe wie unten abgebildet in den frei gewordenen Hohlräumen ablegen. Hoffentlich geht ihm dabei nicht die Puste aus. Alle Roboter müssen sich am Ende wieder im oberen Eingangsstollen befinden.



<<< Zurück zu Level 8 | **Level 9** | Weiter zu Level 10 >>>

Alle Arbeitsaufträge in diesem Namensraum basieren auf den Materialien von Schaller/Zechnall zur Informatikfortbildung Baden-Württemberg 2016 und stehen unter einer CC-BY-SA-NC Lizenz.