Schleifen und Boole‘sche Arrays

- Lottozahlen
- Beispiele für Schleifen
- Abweisende Schleifen
- Lottozahlen mit großer Schleife...
  - ... und Hilfsvariablen
  - ... und Boole‘schem Array

Aufgabenstellung: 6 Lottozahlen ermitteln

- Gesucht ist ein Programm, das sechs (verschiedene) Lottozahlen „6 aus 49“ ermitteln

- Hilfestellung
  Der folgende Ausdruck liefert eine ganze Zahl im Bereich 1 bis 49:

```
(int) (Math.random() * 49) + 1
```

- Nach der Zuweisung

```
int zufälligeZahl = (int) (Math.random() * 49) + 1;
```

  gilt also: 1 ≤ zufälligeZahl ≤ 49

6 Lottozahlen erzeugen und ausgeben

```
println("Das Ziehungsgerät ist in einwandfreiem Zustand.");

println((int) (Math.random() * 49) + 1);
println((int) (Math.random() * 49) + 1);
println((int) (Math.random() * 49) + 1);
println((int) (Math.random() * 49) + 1);
println((int) (Math.random() * 49) + 1);
println((int) (Math.random() * 49) + 1);

println("Die Gewinner werden benachrichtigt.");
```

Was ist das Problem mit dieser Lösung?

Vermeiden doppelter (Lotto-)Zahlen 1/4

Idee

1. Wir ziehen eine Zufallszahl

2. Wir ziehen eine zweite Zufallszahl...

   .. so oft neu, wie sie gleich der ersten Zahl ist

3. Wir ziehen eine dritte Zufallszahl...

   ... so oft neu, wie sie gleich der ersten *oder* gleich der zweiten Zahl ist

4. Wir ziehen eine vierte Zufallszahl...

Vermeiden doppelter (Lotto-)Zahlen 2/4

Idee

1. Wir ziehen eine Zufallszahl

```
int zahl1 = (int) (Math.random() * 49) + 1;
```

2. Wir ziehen eine zweite Zufallszahl...

   .. so oft neu, wie sie gleich der ersten Zahl ist

```
int zahl2 = (int) (Math.random() * 49) + 1;
while (zahl2 == zahl1)
    zahl2 = (int) (Math.random() * 49) + 1;
```

Leseweise:
Solange (engl. *while*) zahl2 gleich zahl1 ist,
bekommt zahl2 einen neuen zufälligen Wert

Vermeiden doppelter (Lotto-)Zahlen 3/4

Idee

3. Wir ziehen eine dritte Zufallszahl...

   ... so oft neu, wie sie gleich der ersten *oder* gleich der zweiten Zahl ist

```
int zahl3 = (int) (Math.random() * 49) + 1;
while (zahl3 == zahl1 || zahl3 == zahl2)
    zahl3 = (int) (Math.random() * 49) + 1;
```

Leseweise:
Solange zahl3 gleich zahl1 ist *oder* zahl3 gleich zahl2 ist,
bekommt zahl3 einen neuen zufälligen Wert

Vermeiden doppelter (Lotto-)Zahlen 4/4

Konsequent weiter für die vierte und die fünfte Zahl:

```
int zahl4 = (int) (Math.random() * 49) + 1;
while (zahl4 == zahl1 || zahl4 == zahl2 || zahl4 == zahl3)
    zahl4 = (int) (Math.random() * 49) + 1;

int zahl5 = (int) (Math.random() * 49) + 1;
while (zahl5 == zahl1 || zahl5 == zahl2 || zahl5 == zahl3 || zahl5 == zahl4)
    zahl5 = (int) (Math.random() * 49) + 1;
```

Und genauso für die sechste Zahl:

```
int zahl6 = (int) (Math.random() * 49) + 1;
while (zahl6 == zahl1 || zahl6 == zahl2 || zahl6 == zahl3 || zahl6 == zahl4 || zahl6 == zahl5)
    zahl6 = (int) (Math.random() * 49) + 1;
```

Wie lautet die Ausgabe des folgenden Programms

```
int i = 1;
while (i < 10)
    i++;
println(i);
```

Ausgabe:

Wie lautet die Ausgabe des folgenden Programms

```
int i = 1;
while (i < 10)
      i++;
      println(i);

println("Ende");
```

Ausgabe:

Wie lautet die Ausgabe des folgenden Programms

```
int i = 1;
while (i < 10) {
    i++;
    println(i);
}
println("Ende");
```

Die *Anweisungsfolge* zählt als eine einzige Anweisung

Ausgabe:

Wie lautet die Ausgabe des folgenden Programms

```
int summe = 0;
int i = 0;
while (i < 10) {
    summe = summe + readInt();
    i++;
}
println(summe);
```

Wie lautet die Ausgabe des folgenden Programms

```
int summe = 0;
int eingabe = readInt();
while (summe != 0) {
    summe = summe + eingabe;
    eingabe = readInt();
}
println(summe);
```

Schleifen in Algorithmen: Ablauf

```
int i = 1;
while (i < 10)
    i++;
println(i);
```

1. Die Bedingung zu Beginn der Schleife wird ausgewertet
   - Ist die Bedingung erfüllt, dann wird der Schleifenrumpf (eine Anweisung) durchlaufen
     Danach wird das Programm am Beginn der Schleife fortgesetzt
   - Ist die Bedingung nicht erfüllt, dann wird der Schleifenrumpf übersprungen

- Es handelt sich um eine *abweisende Schleife*:
  Ist die Bedingung beim ersten Versuch nicht erfüllt, wird der Schleifenrumpf
  nicht ein einziges Mal
  durchlaufen

Die Bedingung in abweisenden Schleifen

```
int i = 1;
while (i < 10)
    i++;
println(i);
```

Es sind drei Fälle möglich

1. Bereits die erste Auswertung der Bedingung liefert *false*
2. Die wiederholte Auswertung der Bedingung liefert eine endliche Folge von
   true, *true*, ..., *true*, *false*
   von Ergebnissen
3. Die wiederholte Auswertung der Bedingung ergibt immer den Wert *true*

Im 3. Fall spricht man von einer *Endlosschleife*, das ist erlaubt, aber selten sinnvoll (und gewollt)

Lottozahlen, etwas eleganter

So viele Zufallszahlen wollen wir bestimmen

```
int anzahlDerKugeln = 0;

while (anzahlDerKugeln < 6) {

    int zufallsZahl = (int) (Math.random() * 49) + 1;

    println(zufallsZahl);

    anzahlDerKugeln = anzahlDerKugeln + 1;

}
```

Das Problem: doppelte Zahlen...

Korrekte Lösung: bereits gezogene Lottozahlen „merken“

```
int anzahlDerKugeln = 0;

while (anzahlDerKugeln < ANZAHL_LOTTOZAHLEN) {

    int zufallsZahl = (int) (Math.random() * 49) + 1;

    while (zahl1 == zufallsZahl
            ||  zahl2 == zufallsZahl
            ||  zahl3 == zufallsZahl
            ||  zahl4 == zufallsZahl
            ||  zahl5 == zufallsZahl)
        zufallsZahl = (int) (Math.random() * 49) + 1;

    anzahlDerKugeln = anzahlDerKugeln + 1;

}
```

Frage: Welchen Typ und welchen Wert haben zahl1, zahl2 usw.?

Hier müssen wir uns noch die aktuelle zufallsZahl in der nächsten freien Variable merken

Korrekte Lösung: bereits gezogene Lottozahlen „merken“

```
int zahl1 = 0, zahl2 = 0, zahl3 = 0, zahl4 = 0, zahl5 = 0;

int anzahlDerKugeln = 0;

while (anzahlDerKugeln < ANZAHL_LOTTOZAHLEN) {

    int zufallsZahl = (int) (Math.random() * 49) + 1;

    while (zahl1 == zufallsZahl
            || zahl2 == zufallsZahl
            || zahl3 == zufallsZahl
            || zahl4 == zufallsZahl
            || zahl5 == zufallsZahl)
        zufallsZahl = (int) (Math.random() * 49) + 1;

    anzahlDerKugeln = anzahlDerKugeln + 1;

}
```

zahl1 bis zahl5 müssen ***vereinbart*** werden

zahl1 bis zahl5 müssen einen ***Wert*** bekommen, ***bevor*** man ***lesen***d darauf zugreifen darf. Sonst: Compilerfehler!

Hier müssen wir uns noch die aktuelle zufallsZahl in der nächsten freien Variable merken

zufallsZahl in der „nächsten freien Variable“ merken

```
if (zahl1 == 0)
    zahl1 = zufallsZahl;
else if (zahl2 == 0)
    zahl2 = zufallsZahl;
else if (zahl3 == 0)
    zahl3 = zufallsZahl;
else if (zahl4 == 0)
    zahl4 = zufallsZahl;
else if (zahl5 == 0)
    zahl5 = zufallsZahl;
```

zahl1 bis zahl5 haben alle den Anfangswert 0 bekommen, Lottozahlen sind alle ≥ 1

Die Prüfung, ob $zahl_i$ schon eine Lottozahl enthält, ist einfach

Alternative Möglichkeit:
eine boolean-Markierung, ob eine Zahl schon gezogen wurde

boolean-Markierung, ob eine Zahl schon gezogen wurde

```
// Indizes von 0 bis 49; 0 wird nicht benutzt
boolean[] gezogen = new boolean[50];

int anzahlDerKugeln = 0;

while (anzahlDerKugeln < ANZAHL_LOTTOZAHLEN) {

    int zufallsZahl = (int) (Math.random() * 49) + 1;

    while (gezogen[zufallsZahl])
        zufallsZahl = (int) (Math.random() * 49) + 1;

    gezogen[zufallsZahl] = true;

    anzahlDerKugeln = anzahlDerKugeln + 1;

}
```

ein Array mit einer *boolean*-Variable pro Lottozahl

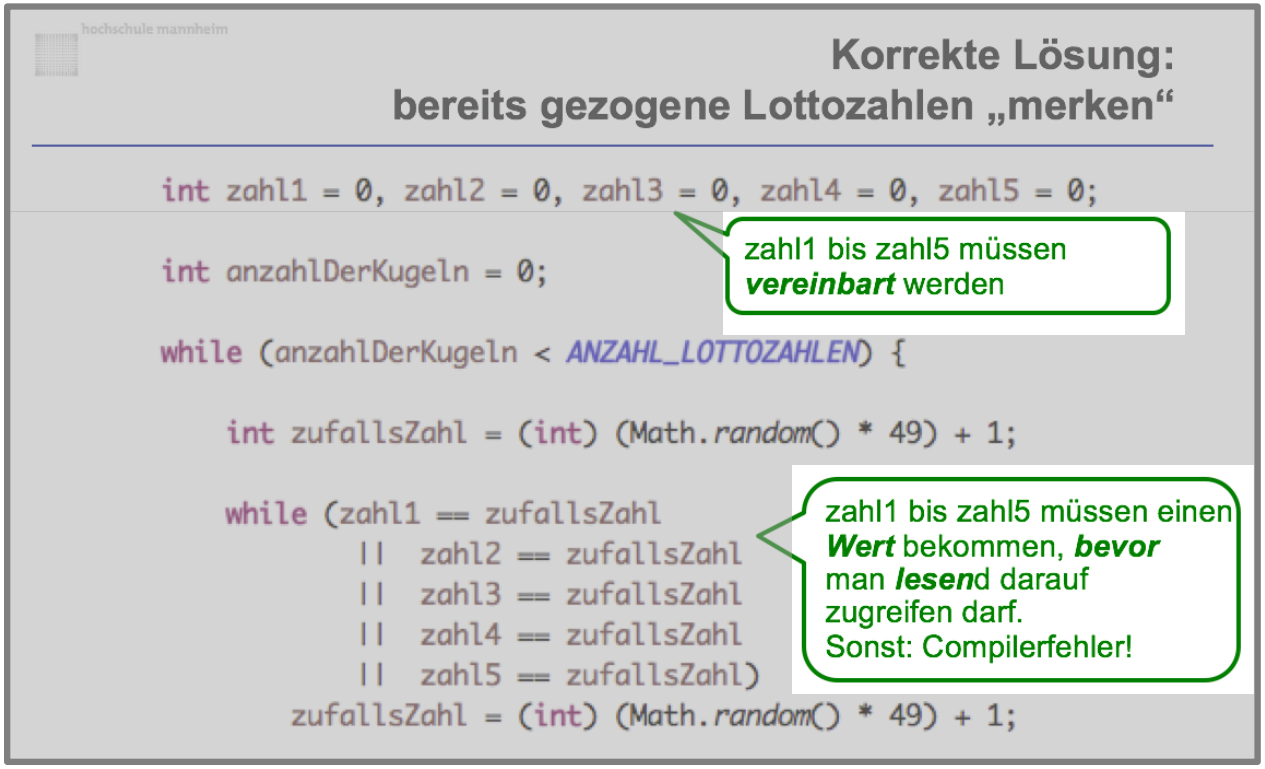
Anfangs sind alle *false*

Markieren derjenigen Zahl, die gerade gezogen wurde

Alternativ: Ausgabe der gezogenen Zahlen nach der Schleife

```
int i = 1;
while (i < 50) {
    if (gezogen[i])
        println(i);
    i++;
}
```

Zur Erinnerung: Variablen-Vereinbarung und -Initialisierung



Was würde sonst passieren?

Beispiel

- Folgendes Beispiel zeigt die Situation:

```
int x;
println( x );
```

- In manchen anderen Programmiersprachen wird das *Bitmuster*, das *zufällig* an der Stelle von *x* *im Speicher* steht, als ganze Zahl (Typ *int*) interpretiert

  -> Es wird „irgendeine“ ganze Zahl ausgegeben

  -> Das ist sehr „ungeschickt“ / gefährlich!

Regelung in Java

- Java ist da etwas strenger:
  Alle Variablen müssen einen definierten Wert bekommen, bevor man ihren Wert lesen kann

```
int x;

println( x );
```

The local variable x may not have been initialized

-> Betrachten Sie das als Hilfestellung!

-> Vermeiden Sie, Variablen mit einem Wert zu versehen, bevor Sie einen sinnvollen Wert verfügbar haben!

-> Vermeiden Sie, Variablen zu vereinbaren, bevor sie benötigt werden!

Beispiel: die schlechte Variante

Obwohl sie noch gar nicht benötigt wird, wird die Variable schon vereinbart UND mit einem (unsinnigen) Wert versehen

Dazwischen wurde vergessen, die Variable vernünftig zu initialisieren

```
int geburtsjahr = 0;

// ... viele Programmzeilen später

println("Sie sind " + (currentYear() - geburtsjahr) + " Jahre alt");
```

Hier wird die Variable mit ihrem unsinnigen Wert verwendet

Beispiel: die bessere Variante

```
int geburtsjahr;

// viele Zeilen später

print("Geben Sie Ihr Geburtsjahr ein: ");
geburtsjahr = readInt();

println("Sie sind " + (currentYear() - geburtsjahr) + " Jahre alt");
```

Die Variable wird erst mit einem Wert versehen, wenn ein vernünftiger Wert verfügbar ist

Würde die „richtige“ Initialisierung vergessen, würde Java hier bei der Verwendung eine Fehlermeldung anzeigen

Beispiel: die gute Variante

Die Variable wird erst vereinbart, wenn sie auch mit einem vernünftigen Wert versehen werden kann

```
print("Geben Sie Ihr Geburtsjahr ein: ");
int geburtsjahr = readInt();

println("Sie sind " + (currentYear() - geburtsjahr) + " Jahre alt");
```

Haben Sie Fragen?

Vorschau

- Es gibt drei weitere Schleifenarten in Java, die wir noch behandeln werden
- Sie gehen auf Nummer sicher, wenn Sie in den Übungen *while*-Schleifen verwenden!
- Wir werden Sie nach *Details* fragen, falls Sie jetzt schon andere Schleifen verwenden!