Bereich: Primitive Datentypen

Überlauf, Zweierkomplement	**Musterlösung**
Package: `de.dhbwka.java.exercise.datatypes`	**Klasse:** `ShortValue`

In der (bei Java verwendeten) Zweierkomplement-Darstellung für ganze Zahlen gibt es kein „Vorzeichen-Bit“, sondern der Wert des höchstwertigen („ersten“, „linken“) Bits wird als negativer Wert interpretiert, im Falle von `short` (16 Bit) hat das höchstwertige Bit also einen Wert von $-2^{15} = -32768_{(10)}$. Alle anderen Bits haben den entsprechenden positiven Wert.
Negative Zahlen sind also solche Zahlen, bei denen das höchstwertige („erste“, „linke“) Bit gesetzt ist.

Die Vorteile dieser Methode sind, dass man im Zweierkomplement ohne Unterscheidung von positiven und negativen Zahlen rechnen kann, und dass es genau eine Darstellung der Zahl `0` (Null) gibt.
Allerdings ergibt sich daraus eben auch, dass bei einem Überlauf (größtmögliche Zahl wird erhöht) ein Übergang zu den negativen Zahlen stattfindet, wie es diese Aufgabe zeigen soll.

Die Binärdarstellung der 16-Bit-Zahl $32767_{(10)}$ ist

2^{-15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
-32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

bzw. kurz:
$0111111111111111_{(2)}$

Das Addieren von $1_{(10)}$
$0000000000000001_{(2)}$

resultiert in der Rechnung

```
  0111111111111111(2) // 32767(10)
+ 0000000000000001(2) //     1(10)
------------------
  1000000000000000(2) // -32768(10)  (-32768(10) + 0(10)).
```

Schreibweisen:
$N_{(10)}$ Dezimalzahl
$N_{(2)}$ Dualzahl/Binärzahl

Kurz gesagt werden beim Zweierkomplement negative Zahlen aus positiven Zahlen erzeugt (Multiplikation mit -1), indem die einzelnen Bits invertiert werden (daher der Name „Komplement“) und anschließend eine 1 addiert wird (damit es keine zwei Nullen gibt), z.B. bei der Zahl $4711_{(10)}$:

```
  0001001001100111(2) // 4711(10)
  1110110110011000(2) // bitweise invertiert
+ 0000000000000001(2) // +1(10)
------------------
  1110110110011001(2) // -4711(10)
```

2^{-15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
-32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
1	1	1	0	1	1	0	1	1	0	0	1	1	0	0	1

„Gegenprobe“:

```
  0110110110011001(2) // 28057(10) (ohne höchstes Bit)
                      // -4711(10) = -32768(10) + 28057(10) ✓
```

```
package de.dhbwka.java.exercise.datatypes;

/**
 * @author DHBW lecturer
 * @version 1.0
 *
 * Part of lectures on 'Programming in Java'.
 * Baden-Wuerttemberg Cooperative State University.
 *
 * (C) 2015 by W. Geiger, T. Schlachter, C. Schmitt, W. Suess
 */
public class ShortValue {

	public static void main(String[] args) {
		short val = 32767;
		System.out.println("val:   "+val);
		val++;
		System.out.println("val+1: "+val);
	}

}
```

Bereich: Primitive Datentypen

Kaufmännisches Runden	**Musterlösung**
Package: `de.dhbwka.java.exercise.datatypes`	**Klasse:** `Round`

```java
package de.dhbwka.java.exercise.datatypes;

/**
 * @author DHBW lecturer
 * @version 1.0
 *
 * Part of lectures on 'Programming in Java'.
 * Baden-Wuerttemberg Cooperative State University.
 *
 * (C) 2016 by W. Geiger, T. Schlachter, C. Schmitt, W. Suess
 */
public class Round {
	public static void main(String[] args) {
		double pi = 3.1415926;  // Naeherung der Kreiszahl Pi
		double e  = 2.7182818;  // Naeherung der Eulerschen Zahl e
		// Positive Zahlen
		System.out.println("Runden positiver Zahlen");
		int piInt = (int) (pi + 0.5);
		int eInt = (int) (e + 0.5);
		System.out.println("Pi ganzzahlig: " + piInt);  // Ausgabe: 3
		System.out.println("e ganzzahlig: " + eInt);    // Ausgabe: 3

		// Naiver Versuch mit negativen Zahlen:
		System.out.println("Runden negativer Zahlen (fehlerhaft)");
		pi = -pi;
		e = -e;
		piInt = (int) (pi + 0.5);
		eInt = (int) (e + 0.5);
		System.out.println("-Pi ganzzahlig: " + piInt); // Ausgabe: -2
		System.out.println("-e ganzzahlig: " + eInt);   // Ausgabe: -2

		// So klappt es negativ wie positiv
		System.out.println("Runden pos. und negativer Zahlen (korrekt)");
		// pi und e sind noch negativ
		piInt = (int) (pi + ( (pi>0) ? 0.5 : -0.5));
		eInt = (int) (e + ( (e>0) ? 0.5 : -0.5));
		System.out.println("-Pi ganzzahlig: " + piInt); // Ausgabe: -3
		System.out.println("-e ganzzahlig: " + eInt);   // Ausgabe: -3
		// pi und e sind positiv machen
		pi = -pi;
		e = -e;
		piInt = (int) (pi + ( (pi>0) ? 0.5 : -0.5));
		eInt = (int) (e + ( (e>0) ? 0.5 : -0.5));
		System.out.println("Pi ganzzahlig: " + piInt);  // Ausgabe: 3
		System.out.println("e ganzzahlig: " + eInt);    // Ausgabe: 3
	}
}
```