KLAUSUR

PROGRAMMIERUNG 1

5. FEBRUAR 2024

Bedingungen der Klausur:

1. Es handelt sich um eine open book Klausur. **Untersagt** sind jedoch alle Arten von Kommunikation mit anderen, auch nicht mit KI-Portalen, wie z.B. Chat GPT, Bing-Suche etc. Es führt bereits zum Ausschluss, wenn Programme zur Kommunikation (E-Mail, Slack, WhatsApp, Signal, …) geöffnet sind bzw. wenn Webseiten zur Kommunikation geöffnet sind (z.B. Chat GBT, …). Schließen Sie vor Klausur also alle entsprechenden Programme und Webseiten!

2. Am Ende der Prüfung: Öffnen Sie den Dateiexplorer/Finder und wechseln Sie in Ihren Workspace. Laden Sie entweder alle ***.java**-Dateien aus dem package **klausur** hoch oder zippen Sie den package-Ordner und laden Sie die zip-Datei in Moodle hoch (bei Aufgabe Klausur1PZ)!

3. Es sind insgesamt 66 Punkte zu erzielen (Teil 1: 19 Pkt., Teil 2: 29 Pkt., Teil 3: 14 Pkt., fehlerfreies Programm: 4 Pkt.).

4. Schreiben Sie Ihre Klassen im package **klausur**!

Notenspiegel:

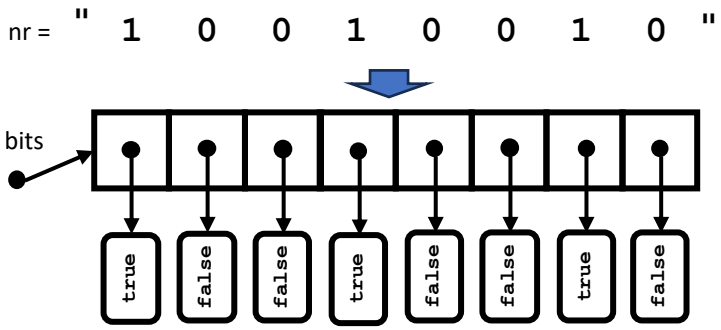
Note	Min	Max
1,0	62,5	66,0
1,3	59,0	62,0
1,7	56,0	58,5
2,0	52,5	55,5
2,3	49,5	52,0
2,7	46,0	49,0
3,0	42,5	45,5
3,3	39,5	42,0
3,7	36,0	39,0
4,0	33,0	35,5
5,0	0	32,5

Teil 1 (Klasse Bit) 19 Punkte

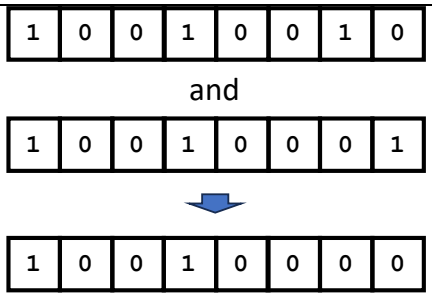
Erstellen Sie eine Klasse **Bit**. Objektvariable ist • **value** vom Typ **boolean.** Die Objektvariablen sind nur innerhalb der Klasse sichtbar!	1 Pkt.
Erstellen Sie für die Klasse **Bit** einen parametrisierten Konstruktor, dem ein Wert für **value** übergeben wird. Initialisieren Sie damit die Objektvariable.	1 Pkt.
Erstellen Sie eine Objektmethode **bitToInt()**. Diese Methode gibt eine **1** zurück, wenn die Objektvariable **value** den Wert **true** hat und eine **0**, wenn der Wert der Objektvariable **false** ist.	2 Pkt.
Erstellen Sie eine Objektmethode **and(Bit b)**, die ein **Bit**-Objekt zurückgibt. Der Wert der Objektvariablen **value** des zurückzugebenden Objektes ergibt sich aus der **&&**-Verknüpfung (Java-UND-Operator) der beiden **value**-Werte des aufrufenden Objektes und **b.** A | B | $A \wedge B$ wahr | wahr | wahr falsch | wahr | falsch wahr | falsch | falsch falsch | falsch | falsch	2 Pkt.
Erstellen Sie eine Objektmethode **or(Bit b)**, die ein **Bit**-Objekt zurückgibt. Der Wert der Objektvariablen **value** des zurückzugebenden Objektes ergibt sich aus der **||**-Verknüpfung (Java-OR-Operator) der beiden **value**-Werte des aufrufenden Objektes und **b.** A | B | $A \vee B$ wahr | wahr | wahr falsch | wahr | wahr wahr | falsch | wahr falsch | falsch | falsch	1 Pkt.
Erstellen Sie eine Objektmethode **isBigger(Bit b)**, die genau dann ein **true** zurückgibt, wenn **value** des aufrufenden Objektes **true** und der Wert von **b false** ist. Ansonsten wird **false** zurückgegeben.	2 Pkt.

Erstellen Sie eine Objektmethode **isEqual(Bit b)**, die genau dann ein **true** zurückgibt, wenn **value** des aufrufenden Objektes gleich dem Wert von **b** ist (also entweder **true** und **true** oder **false** und **false**). Ansonsten wird **false** zurückgegeben.	1 Pkt.
Überschreiben Sie die Methode **toString()** so, dass eine **"1"** zurückgegeben wird, wenn **value** den Wert **true** hat und eine **"0"**, wenn **false**.	2 Pkt.
Erstellen Sie eine **Programmklasse** mit **main()**-Methode. Erzeugen Sie in der **main()-**Methode vier **Bit**-Objekte **b1**, **b2**, **b3** und **b4** mit den Werten **true (b1)**, **true (b2)**, **false (b3)**, **false (b4)**. Erzeugen Sie mithilfe der **toString()**-Methode von **Bit** folgende Konsolenausgaben: **---------- Bit-Objekte ----------** **b1 = 1** **b2 = 1** **b3 = 0** **b4 = 0**	2 Pkt.
Wenden Sie die **and(Bit)-** und **or(Bit)**-Methode jeweils für (**b1,b2**), (**b1, b3**) und (**b3, b4**) (zuerst immer das aufrufende Objekt) an und erzeugen Sie mithilfe der **toString()**-Methode von **Bit** folgende Konsolenausgaben: **---------- Bit and und or ----------** **b1 and b2 = 1** **b1 or b2 = 1** **b1 and b3 = 0** **b1 or b3 = 1** **b3 and b4 = 0** **b3 or b4 = 0**	3 Pkt.
Wenden Sie die **isBigger(Bit)-** und **isEqual(Bit)**-Methode jeweils für (**b1,b2**), (**b1, b3**) und (**b3, b4**) (**>** steht für **isBigger()** und **==** steht für **isEqual()**) an und erzeugen Sie mithilfe der **toString()**-Methode von **Bit** folgende Konsolenausgaben: **---------- Bit Vergleiche ----------** **b1 > b2 ? false** **b1 > b3 ? true** **b3 > b4 ? false** **b1 == b2 ? true** **b1 == b3 ? false** **b3 == b4 ? true**	2 Pkt

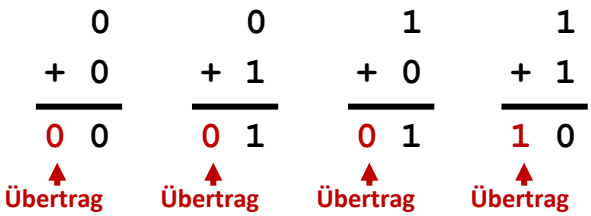
Teil 2 (Klasse Byte) 29 Punkte

Erstellen Sie eine Klasse **Byte**. Objektvariable ist • **bits** vom Typ **Bit[]**. Die Objektvariable ist nur innerhalb der Klasse sichtbar!	1 Pkt.
Erstellen Sie für die Klasse **Byte** einen parameterlosen Konstruktor. Innerhalb des Konstruktors wird das **bits**-Array der Länge **8** erzeugt.	1 Pkt.
Erstellen Sie eine Objektmethode **createByte()**. Diese Methode gibt nichts zurück. • In der Methode wird das **bits**-Array vollständig mit **Bit**-Objekten befüllt. • Die **value**-Werte der einzelnen **Bit**-Objekte werden jeweils zufällig mithilfe der **nextBoolean()**-Methode der Klasse **Random** erzeugt. (**nextBoolean()** gibt zufällig ein **true** oder ein **false** zurück). • Erzeugen Sie sich dazu in der **createByte()**-Methode ein **Random**-Objekt, für das Sie jeweils **nextBoolean()** aufrufen. Die Klasse **Random** muss aus dem **java.util**-Paket importiert werden.	2 Pkt.
Erstellen Sie eine Objektmethode **createByte(String nr)**. Diese Methode gibt nichts zurück. Für den übergebenen String können Sie folgende Annahmen treffen: • Der String hat die Länge **8** • Der String enthält nur die Zeichen **0** und **1** • Beispiele: "10010010", "10010001", "01110001", "01110011" • In der Methode wird das **bits**-Array vollständig mit **Bit**-Objekten befüllt. • Die **value**-Werte der einzelnen **Bit**-Objekte entsprechen dem Zeichen im übergebenen String ('**1**' → **true**, '**0**' → **false**).  *Bit-Objekte*	2 Pkt.

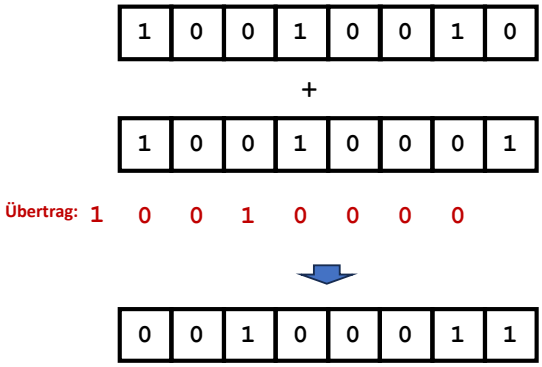
Überschreiben Sie die Methode **toString()** so, dass ein **String** in der folgenden Form zurückgegeben wird (Beispielwerte): **1001 0010** • Nach 4 **Bits** erfolgt ein Leerzeichen	2 Pkt.
Erzeugen Sie in der **main()**-Methode der **Programmklasse** ein Objekt **by1** von **Byte** und befüllen Sie das **bits**-Array mithilfe der **createByte()**-Methode (also zufällig). Erzeugen Sie 5 weite **Byte**-Objekte **by2**, **by3**, **by4**, **by5** und **by6** und befüllen Sie das jeweilige **bits**-Array mithilfe der **createByte(String)**-Methode unter Verwendung folgender Strings: • **by2 → "10010010"** • **by3 → "10010001"** • **by4 → "10010010"** • **by5 → "01110001"** • **by6 → "01110011"** Erzeugen Sie mithilfe der **toString()**-Methode für **Byte** folgende Konsolenausgaben (für **by1** Beispielwerte – zufällig erzeugt): **---------- Byte-Objekte ----------** **by1 = 0011 0100** **by2 = 1001 0010** **by3 = 1001 0001** **by4 = 1001 0010** **by5 = 0111 0001** **by6 = 0111 0011**	3 Pkt.
Erstellen Sie in der Klasse **Byte** eine Objektmethode **isBigger(Byte b).** Diese Methode gibt ein **true** zurück, wenn das aufrufende **Byte**-Objekt einem höheren Byte-Wert entspricht als **b**, also z.B. **"1001 0010"** *isBigger* als **"1001 0001"** oder **"0111 0011"** *isBigger* als **"0111 0001".** Für gleich oder kleiner wird **false** zurückgegeben.	3 Pkt.
Erstellen Sie in der Klasse **Byte** eine Objektmethode **and(Byte b).** Diese Methode gibt ein **Byte**-Objekt zurück. Die Werte des **bits**-Array des zurückzugebenden Objektes ergeben sich durch bitweise **&&**-Verknüpfung (**and(Bit)**-Methode), z.B.: (nächste Seite)	3 Pkt.



Erstellen Sie in der Klasse **Byte** eine Objektmethode **add(Byte b).** Diese Methode gibt ein **Byte**-Objekt zurück. Die Werte des **bits**-Array des zurückzugebenden Objektes ergeben sich durch bitweise „Addition“:



Beispiel:



Verwenden Sie in der **main()**-Methode

- die Methode **isBigger(Byte)** für den Vergleich der Objekte (**by2, by3**), (**by2, by4**) und (**by3, by4**) und erzeugen Sie unter Verwendung der **toString()**-Methode von **Byte** folgende Konsolenausgaben:

  ---------- Byte Vergleiche ----------

  1001 0010 > 1001 0001 ? true
  1001 0010 > 1001 0010 ? false
  1001 0001 > 1001 0010 ? false

6 Pkt.

• die Methode **and(Byte)** für die Operation der Objekte (**by2, by3**) und (**by2, by4**) und erzeugen Sie unter Verwendung der **toString()**-Methode von **Byte** folgende Konsolenausgaben: **---------- Byte and ----------** **1001 0010 and 1001 0001 = 1001 0000** **1001 0010 and 1001 0010 = 1001 0010** • die Methode **add(Byte)** für die Operation der Objekte (**by2, by3**) und (**by5, by6**) und erzeugen Sie unter Verwendung der **toString()**-Methode von **Byte** folgende Konsolenausgaben: **---------- Byte add ----------** **1001 0010 + 1001 0001 = 0010 0011** **0111 0001 + 0111 0011 = 1110 0100**	

Teil 3 (Programmklasse) 14 Punkte

Erzeugen Sie in der **main()**-Methode der **Programmklasse** zwei Byte-Arrays **bya1** und **bya2**. • Das Byte-Array **bya1** hat die Länge **10.** Befüllen Sie es vollständig mit **Byte**-Objekten. Wenden Sie auf die **Byte**-Objekte jeweils die **createByte()**-Methode an (zufällige Werte). • Das Byte-Array **bya2** hat die Länge 2**0.** Befüllen Sie es vollständig mit **Byte**-Objekten. Wenden Sie auf die **Byte**-Objekte jeweils die **createByte()**-Methode an (zufällige Werte).	3 Pkt.
Erstellen Sie in der **Programmklasse** eine statische Methode **printByteArray(Byte[] bya)**. Diese Methode gibt das übergebene **Byte**-Array **bya** auf der Konsole in der folgenden Form aus (Beispiel bya1 – Zufallswerte): **------ Byte-Array der Laenge 10 --------** **0 : 0001 1110** **1 : 1000 0000** **2 : 0010 1000** **3 : 1110 1010** **4 : 0100 0111** **5 : 0010 0101** **6 : 1100 0001**	1 Pkt.

7 : 1010 0111 **8 : 1100 1100** **9 : 0100 0001** • In der obersten Zeile wird also auch immer zunächst die Länge des Arrays mitausgegeben! • Die einzelnen **Byte**-Objekte werden nummeriert (ganz linker Wert in jeder Zeile) beginnend mit **0** (gefolgt von Doppelpunkt).	
Rufen Sie in der **main()**-Methode die **printByteArray()**-Methode für die Byte-Arrays **bya1** und **bya2** auf. Es entstehen folgende Ausgaben (Zufallswerte!): **------ Byte-Array der Laenge 10 --------** **0 : 0001 1110** **1 : 1000 0000** **2 : 0010 1000** **3 : 1110 1010** **4 : 0100 0111** **5 : 0010 0101** **6 : 1100 0001** **7 : 1010 0111** **8 : 1100 1100** **9 : 0100 0001** **------ Byte-Array der Laenge 20 --------** **0 : 1101 0111** **1 : 0110 1000** **2 : 0011 0101** **3 : 1100 0000** **4 : 1001 0010** **5 : 0101 0010** **6 : 1110 1101** **7 : 0110 1100** **8 : 1100 1010** **9 : 1111 1110** **10 : 0101 0101** **11 : 0101 1100** **12 : 1011 1101** **13 : 1011 0101** **14 : 0110 1000** **15 : 1011 0100** **16 : 1001 1000** **17 : 0011 0001** **18 : 1000 1101**	1 Pkt.

19 : 0011 0100	
Erstellen Sie in der **Programmklasse** eine statische Methode **sortByteArray(Byte[] bya)**. Diese Methode gibt ein **Byte**-Array zurück. Das zurückgegebene **Byte**-Array enthält alle **Byte**-Objekte aus **bya** und ist aufsteigend sortiert. Das übergebene **bya** wird nicht sortiert!	7 Pkt.
Rufen Sie in der **main()**-Methode die **sortByteArray()**-Methode für die Byte-Arrays **bya1** und **bya2** auf. Geben Sie das jeweils zurückgegebene Byte-Array mithilfe der **printByteArray()**-Methode auf die Konsole aus. Es entstehen folgende Ausgaben (Beispielwerte - zufällig): **------ Byte-Array der Laenge 10 --------** **0 : 0001 1110** **1 : 0010 0101** **2 : 0010 1000** **3 : 0100 0001** **4 : 0100 0111** **5 : 1000 0000** **6 : 1010 0111** **7 : 1100 0001** **8 : 1100 1100** **9 : 1110 1010** **------ Byte-Array der Laenge 20 --------** **0 : 0011 0001** **1 : 0011 0100** **2 : 0011 0101** **3 : 0101 0010** **4 : 0101 0101** **5 : 0101 1100** **6 : 0110 1000** **7 : 0110 1000** **8 : 0110 1100** **9 : 1000 1101** **10 : 1001 0010** **11 : 1001 1000** **12 : 1011 0100** **13 : 1011 0101** **14 : 1011 1101** **15 : 1100 0000** **16 : 1100 1010**	2 Pkt.

17 : 1101 0111 **18 : 1110 1101** **19 : 1111 1110**	

Zur Kontrolle: Die möglichen Ausgaben (Beispielwerte) könnten sein:

---------- Bit-Objekte ----------

b1 = 1
b2 = 1
b3 = 0
b4 = 0

---------- Bit and und or ----------

b1 and b2 = 1
b1 or b2 = 1
b1 and b3 = 0
b1 or b3 = 1
b3 and b4 = 0
b3 or b4 = 0

---------- Bit Vergleiche ----------

b1 > b2 ? false
b1 > b3 ? true
b3 > b4 ? false
b1 == b2 ? true
b1 == b3 ? false
b3 == b4 ? true

---------- Byte-Objekte ----------

by1 = 1111 0101
by2 = 1001 0010
by3 = 1001 0001
by4 = 1001 0010
by5 = 0111 0001
by6 = 0111 0011

---------- Byte Vergleiche ----------

1001 0010 > 1001 0001 ? true
1001 0010 > 1001 0010 ? false
1001 0001 > 1001 0010 ? false

---------- Byte and ----------

1001 0010 and 1001 0001 = 0000 1001
1001 0010 and 1001 0010 = 0100 1001

---------- Byte add ----------

1001 0010 + 1001 0001 = 0010 0011
0111 0001 + 0111 0011 = 1110 0100

---------- Byte-Array unsortiert ----------

------ Byte-Array der Laenge 10 --------

0 : 0001 1110
1 : 1000 0000
2 : 0010 1000
3 : 1110 1010
4 : 0100 0111
5 : 0010 0101
6 : 1100 0001
7 : 1010 0111
8 : 1100 1100
9 : 0100 0001

------ Byte-Array der Laenge 20 --------

0 : 1101 0111
1 : 0110 1000
2 : 0011 0101
3 : 1100 0000
4 : 1001 0010
5 : 0101 0010
6 : 1110 1101
7 : 0110 1100
8 : 1100 1010
9 : 1111 1110

10 : 0101 0101
11 : 0101 1100
12 : 1011 1101
13 : 1011 0101
14 : 0110 1000
15 : 1011 0100
16 : 1001 1000
17 : 0011 0001
18 : 1000 1101
19 : 0011 0100

---------- Byte-Array sortiert ----------

------ Byte-Array der Laenge 10 --------

0 : 0001 1110
1 : 0010 0101
2 : 0010 1000
3 : 0100 0001
4 : 0100 0111
5 : 1000 0000
6 : 1010 0111
7 : 1100 0001
8 : 1100 1100
9 : 1110 1010

------ Byte-Array der Laenge 20 --------

0 : 0011 0001
1 : 0011 0100
2 : 0011 0101
3 : 0101 0010
4 : 0101 0101
5 : 0101 1100
6 : 0110 1000
7 : 0110 1000
8 : 0110 1100
9 : 1000 1101
10 : 1001 0010
11 : 1001 1000
12 : 1011 0100
13 : 1011 0101

14 : 1011 1101
15 : 1100 0000
16 : 1100 1010
17 : 1101 0111
18 : 1110 1101
19 : 1111 1110

Viel Erfolg!