Grundlagen von Java

Aufgabe 1: Typen und Zuweisungen in Java

Welche der folgenden Java-Anweisungen sind fehlerhaft? Handelt es sich um einen Compiler- oder einen Laufzeitfehler?

Anmerkung: Folgefehler werden jeweils ignoriert.

```
class C {
     public static void main(String[] args) {
          short s = 7;
          int i = -12;
          unsigned int u = +12;
          long l = 123456789;
          float f = 3.1415;
          double d = 3.1415;

          s = i;
          i = s;
          s = 123456789;
          int i2 = (int) l;
          i = 12.4;
          i = 12L;
          i = (int) 12L;

          String str = "Hallo" + "Welt";
          str = args[-3];
          str = "i = " + i;
          str = str - i;

          boolean b = false;
          b = 12L == 12;
     }
}
```

Lösung:

```
public class Aufg1b_Typkorrekt {
   public static void main(String[] args) {
      short s = 7;
      int i = -12;
//    unsigned int u = +12;   // unsigned unbekannt
      long l = 123456789;
      float f = 3.1415F;      //3.1415;   // Typfehler: double statt float!
      double d = 3.1415;

      s = (short) i;          //i;        // Typfehler: int -> short
      System.out.println("s = " + s);          // --> -12
      i = s;
      s = (short) 123456789; //123456789; // Typfehler: int statt short
      System.out.println("s = " + s);      // --> -13035
      int i2 = (int) l;
      System.out.println("i2 = " + i2);   // --> 123456789
      i = (int) 12.4;         //12.4;     // Typfehler: double statt int
      System.out.println("i = " + i);     // --> 12
      i = (int) 12L;          //12L;      // Typfehler: long statt int
      System.out.println("i = " + i);     // --> 12
      i = (int) 12L;

      String str = "Hallo" + "Welt";
//    str = args[-3];         //args[-3]; // Laufzeitfehler:
ArrayIndexOutOfBoundsException
      str = "i = " + i;
      System.out.println("str = " + str); // --> i = 12
      str = str + i;          //str - i;  // Operator - undefiniert für
String x int
      System.out.println("str = " + str); // --> i = 1212

      boolean b = false;
      b = 12L == 12;
      System.out.println("b = " + b);          // --> true
   }
}
```

Zusatz-Aufgabe 2: Installation und Einrichten des JDK

Hinweis: Diese optionale Aufgabe ist zur Einführung dafür gedacht, wie Sie Java auf der Kommandozeile (d.h. mit den Befehlen `javac` und `java`) verwenden können.

Teil 1 – Installation: Das JDK (Java Development Kit) für Java sollte bereits auf den Übungsrechnern im GLI-Labor installiert sein (zumindest in einer Windows-VM). Sofern Sie das JDK auf ihrem eigenen Rechner installieren oder von einer älteren Version updaten wollen, müssen Sie es sich von der Download-Seite (http://www.oracle.com/technetwork/java/javase/downloads/index.html) herunterladen und entsprechend installieren.

Wir benötigen für die Veranstaltung

- Das **JDK (Java Development Kit)**, dort ist der Java-Compiler und weitere Entwicklungs-Tools enthalten. Die JRE (Java Runtime Environment) reicht nicht!
- **mindestens Java Version 8** (oder eine spätere, nachfolgende Version)
- die Java Standard Version (SDK). Die Java Enterprise Version (JEE) ist eine erweiterte Version für verteilte Geschäftsanwendungen, die wir im Wahl-Modul „Fortgeschrittene Java-Programmierung“ benötigen. Jetzt können Sie jedoch erst einmal die vereinfachte SDK-Version verwenden.

Achtung:

- Seit einiger Zeit hat Oracle die Java-Lizenzen angepasst. Auf der Download-Seite von Oracle (s.o.) muss man sich deshalb jeweils persönlich registrieren und bestätigen, dass man sich an die Lizenz-Vereinbarung hält. (Bei kommerzieller Nutzung bestehen Einschränkungen!)
- Auch veralten die „offiziellen“ Java-Versionen von Oracle relativ schnell (Unterstützung bestimmter Versionen teilweise nur 6 Monate).
- Als **Alternative** ist deshalb die Java-Implementierung des **Open JDK Projektes** empfehlenswert. (Im GLI-Labor ist dies implementiert.) Allerdings ist diese Version nicht für „produktionstaugliche“ Anwendungen empfohlen.
    - Siehe https://openjdk.java.net/install/) für Installationshinweise.
    - Das Java 9 JDK kann z.B. unter https://jdk.java.net/java-se-ri/9 heruntergeladen werden. (oder https://jdk.java.net/17/ für Java 17).

Teil 2 – Einrichten des JDK: Damit Sie beim Aufruf des Java-Kommandos `javac` nicht jedesmal den gesamten Pfad angeben müssen (sonst findet Windows das Programm `javac.exe` nicht – dies ist leider standardmäßig nach der Installation des JDK der Fall, so auch auf den Fachbereichsrechern), sollten Sie den Suchpfad für Programme erweitern. Dazu müssen Sie die Umgebungsvariable PATH um den Ordner mit den Java-Programmen (`javac.exe` etc.) erweitern, je nach JDK-Version z.B.

```
C:\Programme\Java\jdk1.9.XYX\bin\
```

Teil 3 – Test: Laden Sie sich das Programm **`Obscure.java`** von der Webseite der Lehrveranstaltung.

Übersetzen und starten Sie das Programm mit dem JDK (durch direkten Aufruf des Java-Compilers).

Was tut dieses Programm?

Anmerkung: In nachfolgenden Übungsaufgaben können Sie auch eine integrierte Entwicklungsumgebung (siehe Aufgabe 3) verwenden.

Lösung zu Obscure.java:

- Webseite der LV in Moodle: https://moodle.hsnr.de/course/view.php?id=7389
- Dieses Programm gibt ein Weihnachtsgedicht aus.
- Das Programm stammt vom „Most Obfuscated C-Programming Contest“ (d.h. einem Wettbewerb für das unleserlichste C-Programm) und wurde vom Dozent entsprechend nach Java übertragen. Es dient als Test, ob das Kompilieren und Starten von Java-Programmen bei Ihnen funktioniert.

```
On the first day of Christmas my true love gave to me
a partridge in a pear tree.

On the second day of Christmas my true love gave to me
two turtle doves
and a partridge in a pear tree.

On the third day of Christmas my true love gave to me
three french hens, two turtle doves
and a partridge in a pear tree.

On the fourth day of Christmas my true love gave to me
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the fifth day of Christmas my true love gave to me
five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the sixth day of Christmas my true love gave to me
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the seventh day of Christmas my true love gave to me
seven swans a-swimming,
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the eigth day of Christmas my true love gave to me
eight maids a-milking, seven swans a-swimming,
six geese a-laying, five gold rings;
```

```
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the ninth day of Christmas my true love gave to me
nine ladies dancing, eight maids a-milking, seven swans a-swimming,
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the tenth day of Christmas my true love gave to me
ten lords a-leaping,
nine ladies dancing, eight maids a-milking, seven swans a-swimming,
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the eleventh day of Christmas my true love gave to me
eleven pipers piping, ten lords a-leaping,
nine ladies dancing, eight maids a-milking, seven swans a-swimming,
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the twelfth day of Christmas my true love gave to me
twelve drummers drumming, eleven pipers piping, ten lords a-leaping,
nine ladies dancing, eight maids a-milking, seven swans a-swimming,
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.
```

Aufgabe 3: Umgang mit Eclipse – Java-Projekte erzeugen

Teil 1 – Installation: Falls Sie eine Entwicklungsumgebung wie z.B. Eclipse auf Ihrem eigenen Rechner ausführen wollen, installieren Sie die Entwicklungsumgebung auf Ihrem Rechner.

> **Anmerkung:** Statt Eclipse können Sie natürlich auch eine andere Entwicklungsumgebung (z.B. NetBeans, IntelliJ) verwenden.

Laden Sie daher von

> http://www.eclipse.org/downloads/

die **Eclipse IDE for Java Developers** (oder **für Java EE Developers** – die erweiterte Version für Java Enterprise) herunter. Achten Sie auf die passende Version (32 oder 64 bit)!

Nach dem Entpacken des zip-Files kann das Programm **eclipse.exe** direkt gestartet werden.

> **Hinweis:** Eclipse wird lediglich in einem beliebigen Ordner entpackt und kann von dort gestartet werden. Allerdings erfolgt dadurch auch keine Eintragung in der Registry und es wird kein Link im Programme-Menü bzw. auf dem Desktop erzeugt – diese müsste man bei Bedarf ggf. selbst selber anlegen.

Teil 2 – Projekt erzeugen: Erstellen Sie ein neues Java-Projekt *("File → New → Java Project").*

- Fügen Sie dem src-Ordner des Projektes (im default-Package) eine neue Java-Klasse hinzu. *("New → Class")*
- Sofern diese Klasse eine passende main-Methode besitzt, können Sie dieses Programm durch Rechtsklick mit der Maus auf die Java-Quelldatei mittels *"Run As → Java Application"* starten.

Sie können dafür z.B. die Datei Obscure.java (siehe Aufgabe 2) oder ein einfaches "Hello World"-Programm nehmen.

Aufgabe 4: Elementare Java Programmierung – Schiffe versenken

Schreiben Sie ein Programm, mit dem Sie gegen den Computer „Schiffe versenken" spielen können.

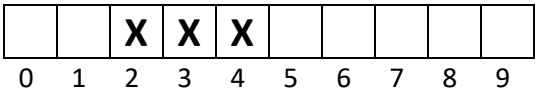
Ziel des Spiels: Versenken Sie alle Schiffe des Gegners mit möglichst wenigen Versuchen. Am Ende erhält der Spieler eine Bewertung, wie gut er war.

Da wir momentan noch keine Kommunikation verwenden können, um mehrere Programme gegeneinander antreten zu lassen, werden wir zunächst gegen den Computer spielen. Ferner müssen wir in diesem frühen Stadium des Semesters auf eine grafische Benutzeroberfläche verzichten, also arbeitet diese Version auf der Kommandozeile.

Setup: Wenn das Spielprogramm gestartet wird, platziert der Computer die Schiffe auf dem Spielfeld. Zur Vereinfachung beschränken wir uns zunächst auf ein eindimensionales Spielfeld, auf welchem das/die Schiffe fest platziert werden, d.h. das Spielfeld kann fest vorgegeben werden.

Falls Sie noch Zeit und Lust haben, können Sie Ihr Programm in Aufgabe 4 entsprechend erweitern.

Wie Sie spielen: Der Computer fordert Sie auf einen Tipp (eine Position bzw. Zelle) abzugeben, den Sie auf der Kommandozeile eingeben. Als Antwort auf Ihren Tipp sehen Sie auf der Kommandozeile als Ergebnis „Vorbei", „Treffer" oder „Versenkt". Wenn das Schiff (bzw. alle Schiffe) versenkt sind, wird ihre Bewertung, d.h. die Anzahl ihrer Versuche ausgegeben.



Teil 1: Erstellen Sie eine Klasse **`SchiffeVersenken1D`**. Erstellen Sie (in dieser Klasse) die folgenden Methoden:

- **`String toString(boolean[]spielFeld)`** liefert eine textuelle Darstellung des Spielfeldes. Diese Funktion kann u.a. zum Testen verwendet werden, um sich die (noch nicht versenkten) Schiffe anzuzeigen.
  - Überlegen Sie sich, wie Sie das Wasser und wie die Schiffe repräsentieren wollen.

- **`int eingabeTipp()`** liest ihren Rateversuch, d.h. eine mögliche Position, von der Kommandozeile ein.

- **`void run()`** implementiert schließlich das Spiel:
  - Zunächst wird das Spielfeld als festes Array erzeugt, d.h. das Schiff auf dem Spielfeld platziert.
  - Im eigentlichen Spielablauf wird der Benutzer jeweils zur Eingabe eines Rateversuches aufgefordert, bis das Schiff versenkt ist.

Fügen Sie schließlich die folgende **`main`**-Funktion zu ihrer Klasse hinzu: (Anmerkung: Später werden wir lernen, warum wir nicht einfach alles direkt in der main-Funktion ausführen können.)

```
public static void main(String[] args) {
      SchiffeVersenken1D spiel = new SchiffeVersenken1D();
      spiel.run();
}
```

Teil 2: Testen Sie Ihr Programm.

Lösung: Aufg4_SchiffeVersenken1D.java

```
import java.util.Scanner;

public class Aufg4_SchiffeVersenken1D {

   /**
    * Erzeugt eine textuelle Darstellung des Spielfeldes,
    * wobei Schiffsteile mit o und Wasserfelder mit _ dargestellt werden.
    *
    * @param feld - das Spielfeld
    * @return textuelle Darstellung.
    */
   String toString(boolean[] feld) {
      String ausgabe = "";
      for (int i = 0; i < feld.length; i++) {
            String c;
            if (feld[i])
                  c = "o";    // Schiff
            else
                  c = "_";    // Wasser
            ausgabe += c + ' ';
      }
      return ausgabe;
   }
```

```
   /**
    * Liest eine Zahl (geratener Tipp) vom Terminal ein.
    * @return - der Tipp
    */
   int eingabeTipp() {
      System.out.print("Geben Sie eine Zahl ein: ");
      Scanner scanner = new Scanner( System.in );
      int eingabe = scanner.nextInt();

      return eingabe;
   }


   /**
    * Startet das Schiffeversenken-Programm
    * @param args - übergebene Aufrufargumente an das Programm
    */
   public static void main(String[] args) {
      Aufg4_SchiffeVersenken1D spiel = new Aufg4_SchiffeVersenken1D();
      spiel.run();
   }

   void run() {
      // Erzeuge ein Spielfeld der Groesse 10 mit einem Schiff d. Groesse 3
      //          - false = Wasser
      //          - true      = SCHIFF
      // (Beachte: Hier ist das Schiff fest positioniert,
      //           eine Zufallswahl wird in
      //           Methode setup() - siehe SchiffeVersenken1DExtendend –
      //           realisiert.)
      final int SCHIFFS_GROESSE = 3;
      boolean[] feld = { false, false, true, true, true, false, false,
false, false, false };

      int anzahlVersuche = 0;
      int anzahlTreffer = 0;

      boolean lebt = true;    // Spiel noch "am Leben"?
      while (lebt == true) {
            int tipp = eingabeTipp();
            anzahlVersuche++;

            boolean ergebnis = pruefeTipp(feld, tipp);
            String ausgabe = "Vorbei";
            if (ergebnis == true) {
                  ausgabe = "Treffer";
                  anzahlTreffer++;
                  feld[tipp] = false;

                  if (anzahlTreffer == SCHIFFS_GROESSE) {
                        ausgabe = "Versenkt";
                        lebt = false;     // Ende
                  }
            }

            System.out.println(ausgabe);
            // System.out.println("Spielfeld: " + toString(feld));     //
Schummeln :-)
      }
```

```
      System.out.println("Sie haben " + anzahlVersuche + " Versuche
benötigt.");
      System.out.println("Spielfeld: " + toString(feld));
   }


   /**
    * Prueft, ob der Spieler ein Schiff getroffen hat.
    *
    * @param feld
    * @param tippPos
    * @return
    */
   boolean pruefeTipp(boolean[] feld, int tippPos) {
      // Test auf falsche Eingabe
      if (feld == null)
            return false;
      if (tippPos < 0 || tippPos > feld.length)
            return false;

      // kürzer: return feld[tippPos]
      if (feld[tippPos] == true) {
            return true;
      } else {
            return false;
      }
   }
}
```

Zusatz-Aufgabe 5 (Nicht ausgelastet?)

Erweiterung1 : Erweitern Sie Ihr Programm um die Möglichkeit, ein Schiff zufällig vom Computer auf dem Spielfeld platzieren zu lassen, indem Sie die folgende **`setup`**-Methode erstellen:

- **`boolean[] setup()`** erzeugt ein (eindimensionales) Spielfeld der Größe 10, auf dem an zufälliger Position ein Schiff der Größe 3 positioniert ist.
  Erstellen Sie dazu die folgenden Hilfsfunktionen:
  - **`boolean[] erzeugeSpielfeld(int feldGroesse)`** erzeugt ein eindimensionales leeres Spielfeld der angegeben Größe (ohne Schiffe). Hinweise:
    - Ein neues Feld können Sie mit **`new boolean[<Länge>]`** erzeugen.
    - Überlegen Sie sich, wie Sie das Wasser und wie die Schiffe repräsentieren wollen.
  - **`boolean setSchiff(boolean spielFeld[], int pos, int schiffsGroesse)`** positioniert ein Schiff bestehend aus **`schiffsGroesse`** Feldern ab Position **`pos`** auf dem Spielfeld. Falls dies nicht möglich ist (wann tritt hier ein Fehler auf?), soll das Spielfeld unverändert bleiben und **`false`** als Fehlercode zurückgeliefert werden.

Lösung: Aufg5-1:_Aufg5a_SchiffeVersenken1D_Extended.java

```
import java.util.Scanner;

public class Aufg5a_SchiffeVersenken1D_Extended {

   /**
    * Erzeugt eine textuelle Darstellung des Spielfeldes,
    * wobei Schiffsteile mit o und Wasserfelder mit _ dargestellt werden.
    *
    * @param feld - das Spielfeld
    * @return textuelle Darstellung.
    */
   String toString(boolean[] feld) {
      String ausgabe = "";
      for (int i = 0; i < feld.length; i++) {
            String c;
            if (feld[i])
                  c = "o";    // Schiff
            else
                  c = "_";    // Wasser
            ausgabe += c + ' ';
      }
      return ausgabe;
   }

   /**
```

```
     * Liest eine Zahl (geratener Tipp) vom Terminal ein.
     * @return - der Tipp
     */
    int eingabeTipp() {
       System.out.print("Geben Sie eine Zahl ein: ");
       Scanner scanner = new Scanner( System.in );
       int eingabe = scanner.nextInt();

       return eingabe;
    }



    /**
     * Startet das Schiffeversenken-Programm
     * @param args - übergebene Aufrufargumente an das Programm
     */
    public static void main(String[] args) {
       Aufg5a_SchiffeVersenken1D_Extended spiel =
                          new Aufg5a_SchiffeVersenken1D_Extended();
       spiel.run();
    }

    void run() {
       boolean[] feld = setup();

       int anzahlVersuche = 0;
       int anzahlTreffer = 0;

       boolean lebt = true;    // Spiel noch "am Leben"?
       while (lebt == true) {
             int tipp = eingabeTipp();
             anzahlVersuche++;

             boolean ergebnis = pruefeTipp(feld, tipp);
             String ausgabe = "Vorbei";
             if (ergebnis == true) {
                   ausgabe = "Treffer";
                   anzahlTreffer++;
                   feld[tipp] = false;

                   if (anzahlTreffer == SCHIFFS_GROESSE) {
                         ausgabe = "Versenkt";
                         lebt = false;     // Ende
                   }
             }

             System.out.println(ausgabe);
             // System.out.println("Spielfeld: " + toString(feld));      //
Schummeln :-)
       }

       System.out.println("Sie haben " + anzahlVersuche + " Versuche
benötigt.");
       System.out.println("Spielfeld: " + toString(feld));
    }


    /**
     * Prueft, ob der Spieler ein Schiff getroffen hat.
```

```
     *
     * @param feld
     * @param tippPos
     * @return
     */
    boolean pruefeTipp(boolean[] feld, int tippPos) {
       // Test auf falsche Eingabe
       if (feld == null)
             return false;
       if (tippPos < 0 || tippPos > feld.length)
             return false;

       // kurz: return feld[tippPos]
       if (feld[tippPos] == true) {
             return true;
       } else {
             return false;
       }
    }



    /**
     * Erzeuge ein leeres Spielfeld OHNE Schiffe (false == Wasser)
     * @param feldGroesse - Anzahl der Felder
     * @return das leere Spielfeld
     */
    boolean[] erzeugeSpielfeld(int feldGroesse) {
       boolean[] feld = new boolean[feldGroesse];
       for (int i = 0; i < feldGroesse; i++) {
             feld[i] = false;  // kein Schiff
       }
       return feld;
    }

    /**
     * Positioniere ein Schiff in das Spielfeld, sofern moeglich
     *
     * @param spielFeld - Das Spielfeld
     * @param pos  - Die Startposition des Schiffes
     * @param schiffsGroesse - Die Groesse des Schiffes (Anzahl belegter
Felder)
     * @return - true, falls das Schiff positioniert werden kann.
     */
    boolean setSchiff(boolean spielFeld[], int pos, int schiffsGroesse) {
       // Teste, ob alles OK ist.
       if (spielFeld == null)
             return false;
       if ((pos < 0) || (schiffsGroesse <= 0)
                     || (pos + schiffsGroesse > spielFeld.length))
             return false;     // Fehler

       // jetzt koennen wir das Schiff positionieren
       for (int i = pos; i < pos + schiffsGroesse; i++) {
             spielFeld[i] = true;    // Schiff
       }
       return true;
    }

    /**
```

```
    * Erzeuge ein Spielfeld der Groesse 10 mit einem Schiff der Groesse 3
    * return: Das Spielfeld
    */
   final int FELD_GROESSE = 10;
   final int SCHIFFS_GROESSE = 3;
   boolean[] setup() {
      boolean[] feld = erzeugeSpielfeld(FELD_GROESSE);

      // erzeuge Schiff (Groesse 3)
      int startPos =
            (int)(Math.random() * (FELD_GROESSE - SCHIFFS_GROESSE + 1));
      setSchiff(feld, startPos, SCHIFFS_GROESSE);

      return feld;
   }

}
```

Erweiterung2: Erweitern Sie Ihr Programm um die Möglichkeit, mehrere Schiffe mit unterschiedlicher Größe zu platzieren.

Bei dieser Gelegenheit bietet es sich auch gleich an, das Spielfeld nicht als ein Feld von `boolean`-Werten zu verwalten, sondern einen expliziten Aufzählungstyp zu verwenden:

```
enum Feld {WASSER, SCHIFF, TREFFER};
```

Lösung: Aufg5-2:_Aufg5b_SchiffeVersenken1D_Extended_mitEnum.java

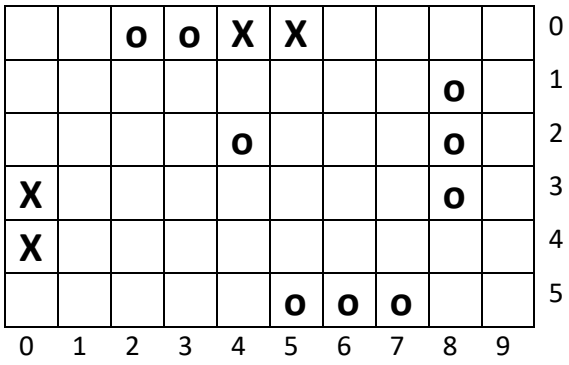
```
import java.util.Scanner;

public class Aufg5b_SchiffeVersenken1D_Extended_mitEnum {

   /**
    * Erzeugt eine textuelle Darstellung des Spielfeldes,
    * wobei Schiffsteile mit o bzw. X (getroffen) und Wasserfelder
    * mit _ dargestellt werden.
    *
    * @param feld - das Spielfeld
    * @return textuelle Darstellung.
    */
   String toString(Feld[] feld) {
      String ausgabe = "";
      for (int i = 0; i < feld.length; i++) {
            String c;
            if (feld[i] == Feld.SCHIFF)
                  c = "o";    // Schiff
            else if (feld[i] == Feld.TREFFER)
                  c = "X";    // Treffer
```

```
            else //if (feld[i] == Feld.WASSER)
                  c = "_";    // Wasser
            ausgabe += c + ' ';
      }
      return ausgabe;
   }

   /**
    * Liest eine Zahl (geratener Tipp) vom Terminal ein.
    * @return - der Tipp
    */
   int eingabeTipp() {
      System.out.print("Geben Sie eine Zahl ein: ");
      Scanner scanner = new Scanner( System.in );
      int eingabe = scanner.nextInt();

      return eingabe;
   }



   /**
    * Startet das Schiffeversenken-Programm
    * @param args - übergebene Aufrufargumente an das Programm
    */
   public static void main(String[] args) {
      Aufg5b_SchiffeVersenken1D_Extended_mitEnum spiel =
                  new Aufg5b_SchiffeVersenken1D_Extended_mitEnum();
      spiel.run();
   }

   void run() {
      Feld[] feld = setup();

      int anzahlVersuche = 0;
      int anzahlTreffer = 0;

      boolean lebt = true;    // Spiel noch "am Leben"?
      while (lebt == true) {
            int tipp = eingabeTipp();
            anzahlVersuche++;

            Feld ergebnis = pruefeTipp(feld, tipp);
            String ausgabe = "Vorbei";
            if (ergebnis == Feld.WASSER) {
                  ausgabe = "Vorbei";
            }
            else if (ergebnis == Feld.SCHIFF) {
                  ausgabe = "Treffer";
                  anzahlTreffer++;
                  feld[tipp] = Feld.TREFFER;

                  if (pruefeVersenkt(feld)) {
                        ausgabe = "Versenkt";
                        lebt = false;     // Ende
                  }
            } else /*if (ergebnis == Feld.TREFFER)*/ {
                  ausgabe = "Was soll das? Das Feld haben Sie doch schon
getroffen.";
            }
```

```
            System.out.println(ausgabe);
            // System.out.println("Spielfeld: " + toString(feld));     //
Schummeln :-)
      }

      System.out.println("Sie haben " + anzahlVersuche + " Versuche
benötigt.");
      System.out.println("Spielfeld: " + toString(feld));
   }


   /**
    * Prueft, ob der Spieler ein Schiff getroffen hat.
    *
    * @param feld
    * @param tippPos
    * @return
    */
   Feld pruefeTipp(Feld[] feld, int tippPos) {
      // Test auf falsche Eingabe
      if (feld == null)
            return Feld.WASSER;
      if (tippPos < 0 || tippPos > feld.length)
            return Feld.WASSER;

      return feld[tippPos];
   }

   /**
    * Prueft, ob der Spieler alle Schiffe versenkt hat.
    *
    * @param feld
    * @return
    */
   boolean pruefeVersenkt(Feld[] feld) {
      // Test auf falsche Eingabe
      if (feld == null)
            return false;

      for (int i = 0; i < feld.length; i++) {
            if (feld[i] == Feld.SCHIFF)
                  return false;     // es gibt noch nicht getroffene (Teile
von) Schiffen
      }
      return true;      // alles WASSER oder VERSENKT
   }


   /*
    * Erzeuge ein leeres Spielfeld OHNE Schiffe (false == Wasser)
    * @param feldGroesse - Anzahl der Felder
    * @return das leere Spielfeld
    */
   Feld[] erzeugeSpielfeld(int feldGroesse) {
      Feld[] feld = new Feld[feldGroesse];
      for (int i = 0; i < feldGroesse; i++) {
            feld[i] = Feld.WASSER;  // kein Schiff
      }
      return feld;
```

```
    }

    /**
     * Positioniere ein Schiff in das Spielfeld, sofern moeglich
     *
     * @param spielFeld - Das Spielfeld
     * @param pos  - Die Startposition des Schiffes
     * @param schiffsGroesse - Die Groesse des Schiffes (Anzahl belegter
Felder)
     * @return - true, falls das Schiff positioniert werden kann.
     */
    boolean setSchiff(Feld spielFeld[], int pos, int schiffsGroesse) {
       // Teste, ob alles OK ist.
       if (spielFeld == null)
             return false;
       if ((pos < 0) || (schiffsGroesse <= 0)
                   || (pos + schiffsGroesse > spielFeld.length))
             return false;     // Fehler

       // jetzt koennen wir das Schiff positionieren
       for (int i = pos; i < pos + schiffsGroesse; i++) {
             spielFeld[i] = Feld.SCHIFF;   // Schiff
       }
       return true;
    }

    /**
     * Erzeuge ein Spielfeld der Groesse 10 mit einem Schiff der Groesse 3
     * return: Das Spielfeld
     */
    enum Feld {WASSER, SCHIFF, TREFFER};
    final int FELD_GROESSE = 10;
    final int SCHIFFS_GROESSE = 3;
    Feld[] setup() {
       Feld[] feld = erzeugeSpielfeld(FELD_GROESSE);

       // erzeuge Schiff (Groesse 3)
       int startPos =
              (int) (Math.random() * (FELD_GROESSE - SCHIFFS_GROESSE + 1));
       setSchiff(feld, startPos, SCHIFFS_GROESSE);

       //System.out.println("setup: startpos = " + startPos);     //
Schummeln :-)

       return feld;
    }
}
```

Immer noch nicht ausgelastet? Wie wäre es, wenn Sie statt eines eindimensionalen Spielfelds ein zweidimensionales Spielfeld verwenden? Für die geratene Position müssten Sie dann jeweils Zeile und Spalte einlesen (oder könnte man das auch anders lösen?)



(Beispieldarstellung: Leer = Wasser, **o** = Schiff (noch nicht getroffen), **X** = Treffer)

```
import java.util.Scanner;

public class Aufg5b_SchiffeVersenken2D_Extended_mitEnum {


      /**
       * Erzeugt eine textuelle Darstellung des Spielfeldes,
       * wobei Schiffsteile mit o bzw. X (getroffen) und Wasserfelder
mit _ dargestellt werden.
       *
       * Beispiel-Ausgabe:
       *     +----------------------+
      *      | _ _ o o X X _ _ _ _ |0
      *      | _ _ _ _ _ _ _ _ o _ |1
      *      | _ _ _ _ o _ _ _ o _ |2
      *      | X _ _ _ _ _ _ _ o _ |3
      *      | X _ _ _ _ _ _ _ _ _ |4
      *      | _ _ _ _ _ o o o _ _ |5
      *      +----------------------+
      *        0 1 2 3 4 5 6 7 8 9
       *
       * @param feld - das Spielfeld
       * @return textuelle Darstellung.
       */
      String toString(Feld[][] feld) {
            String spaces = "    ";         // Abstand links vom Rand

            String linie_vertikal = spaces + "+-";     //
Begrenzungslinie
            for (int spalte = 0; spalte < feld[0].length; spalte++) {
                  linie_vertikal += "--";
            }
            linie_vertikal += "+\n";

            String ausgabe = "\n" + linie_vertikal;
```

```
            for (int zeile = 0; zeile < feld.length; zeile++) {
                  ausgabe += spaces + "| ";
                  for (int spalte = 0; spalte < feld[0].length;
spalte++) {
                        String c;
                        if (feld[zeile][spalte] == Feld.SCHIFF)
                              c = "o";   // Schiff
                        else if (feld[zeile][spalte] == Feld.TREFFER)
                              c = "X";   // Treffer
                        else //if (feld[zeile][spalte] == Feld.WASSER)
                              c = "_";   // Wasser
                        ausgabe += c + ' ';
                  }
                  // Zeilenumbruch
                  ausgabe += "|" + zeile + "\n";
            }

            ausgabe += linie_vertikal;

            ausgabe += spaces + "  ";
            for (int spalte = 0; spalte < feld[0].length; spalte++) {
                  ausgabe += spalte + " ";
            }
            ausgabe += "\n";

            return ausgabe;
      }

      /**
       * Liest einen Tipp (geratene Zeile & Spalte) vom Terminal ein.
       * @return - der Tipp als Feld von 2 Zahlen [zeile, spalte]
       *                (das ist sehr unsauber - besser: Rückgabe
eines Paar-Objektes statt Array!)
       */
      int[] eingabeTipp() {
            Scanner scanner = new Scanner( System.in );

            System.out.print("Geben Sie die Zeile ein: ");
            int zeile = scanner.nextInt();
            System.out.print("Geben Sie die Spalte ein: ");
            int spalte = scanner.nextInt();

            int[] ergebnis = new int[2];    // Array von 2 ganzen
Zahlen
            ergebnis[0] = zeile;       // das ist sehr unsauber !
            ergebnis[1] = spalte;
            return ergebnis;
      }



      /**
       * Startet das Schiffeversenken-Programm
       * @param args - übergebene Aufrufargumente an das Programm
       */
```

```
	public static void main(String[] args) {
		Aufg5b_SchiffeVersenken2D_Extended_mitEnum spiel = new
Aufg5b_SchiffeVersenken2D_Extended_mitEnum();
		spiel.run();

	}

	void run() {
		Feld[][] feld = setup();

		int anzahlVersuche = 0;
		int anzahlTreffer = 0;

		boolean lebt = true;  // Spiel noch "am Leben"?
		while (lebt == true) {
			System.out.println("Spielfeld: " + toString(feld));
	// Schummeln :-)

			int[] tipp = eingabeTipp();
			anzahlVersuche++;

			Feld ergebnis = pruefeTipp(feld, tipp);
			String ausgabe = "Vorbei";
			if (ergebnis == Feld.WASSER) {
				ausgabe = "Vorbei";
			}
			else if (ergebnis == Feld.SCHIFF) {
				ausgabe = "Treffer";
				anzahlTreffer++;

				int tippZeile = tipp[0];
				int tippSpalte = tipp[1];
				feld[tippZeile][tippSpalte] = Feld.TREFFER;

				if (pruefeVersenkt(feld)) {
					ausgabe = "Versenkt";
					lebt = false;   // Ende
				}
			} else /*if (ergebnis == Feld.TREFFER)*/ {
				ausgabe = "Was soll das? Das Feld haben Sie
doch schon getroffen.";
			}

			System.out.println(ausgabe);
		}

		System.out.println("Sie haben " + anzahlVersuche + "
Versuche benötigt.");
		System.out.println("Spielfeld: " + toString(feld));
	}


	/**
	 * Prueft, ob der Spieler ein Schiff getroffen hat.
	 *
```

```
	 * @param feld
	 * @param tippPos
	 * @return
	 */
	Feld pruefeTipp(Feld[][] feld, int[] tippPos) {
		// Test auf falsche Eingabe
		if (feld == null)
			return Feld.WASSER;
		if (tippPos == null || tippPos.length != 2)
			return Feld.WASSER;
		int zeile = tippPos[0];		// unsauber !
		int spalte = tippPos[1];
		if (zeile < 0 || zeile >= feld.length)	// Zeilen-Index
korrekt
			return Feld.WASSER;
		if (spalte < 0 || spalte >= feld[0].length)	//
Spalten-Index korrekt
			return Feld.WASSER;

		return feld[zeile][spalte];
	}

	/**
	 * Prueft, ob der Spieler alle Schiffe versenkt hat.
	 *
	 * @param feld
	 * @return
	 */
	boolean pruefeVersenkt(Feld[][] feld) {
		// Test auf falsche Eingabe
		if (feld == null)
			return false;

		for (int zeile = 0; zeile < feld.length; zeile++) {
			for (int spalte = 0; spalte < feld[0].length;
spalte++) {
			if (feld[zeile][spalte] == Feld.SCHIFF)
				return false;	// es gibt noch nicht
getroffene (Teile von) Schiffen
			}
		}
		return true;	// alles WASSER oder VERSENKT
	}


	/*
	 * Erzeuge ein leeres Spielfeld OHNE Schiffe (false == Wasser)
	 * @param anzahlZeilen - Anzahl der Zeilen (Größe des
Spielfeldes)
	 * @param anzahlSpalten - Anzahl der Spalten
	 * @return das leere Spielfeld (überall ist Wasser)
	 */
	Feld[][] erzeugeSpielfeld(int anzahlZeilen, int anzahlSpalten)
{
		Feld[][] feld = new Feld[anzahlZeilen][anzahlSpalten];
```

```
            for (int zeile = 0; zeile < anzahlZeilen; zeile++) {
                  for (int spalte = 0; spalte < anzahlSpalten;
spalte++) {
                        // Wichtig: alle Spielfelder explizit
initialisieren
                        // (anderenfalls haben sie den Default-Wert
null!)
                        feld[zeile][spalte] = Feld.WASSER;    // kein
Schiff
                  }
            }
            return feld;
      }

      /**
       * Positioniere ein Schiff in das Spielfeld, sofern moeglich
       *
       * @param spielFeld - Das Spielfeld
       * @param posX  - Die Startposition des Schiffes (Spalte)
       * @param posY  - Die Startposition des Schiffes (Zeile)
       * @param schiffsBreite - Die Breite des Schiffes (Anzahl
belegter Felder in y-Richtung)
       * @param schiffsHoehe - Die Hoehe des Schiffes (Anzahl
belegter Felder in x-Richtung)
       * @return - true, falls das Schiff positioniert werden kann.
       */
      boolean setSchiff(Feld spielFeld[][], int posX, int posY, int
schiffsBreite, int schiffsHoehe) {
            // Teste, ob alles OK ist.
            if (spielFeld == null)
                  return false;
            if ((posY < 0) || (schiffsHoehe <= 0) || (posY +
schiffsHoehe > spielFeld.length))
                  return false;   // Fehler
            if ((posX < 0) || (schiffsBreite <= 0) || (posX +
schiffsBreite > spielFeld[0].length))
                  return false;   // Fehler

            // Prüfe, ob Felder oder Nachbarn bereits belegt sind
            for (int zeile = Math.max(posY - 1, 0);
                  zeile < Math.min(posY + schiffsHoehe + 1,
spielFeld.length-1);
                  zeile++) {
                  for (int spalte = Math.max(posX - 1, 0);
                        spalte < Math.min(posX + schiffsBreite + 1,
spielFeld[0].length);
                        spalte++) {
                        if (spielFeld[zeile][spalte] == Feld.SCHIFF)
                              return false;   // bereits belegt
                  }
            }

            // jetzt koennen wir das Schiff positionieren
            for (int zeile = posY; zeile < posY + schiffsHoehe;
zeile++) {
```

```
                for (int spalte = posX; spalte < posX +
schiffsBreite; spalte++) {
                      spielFeld[zeile][spalte] = Feld.SCHIFF;    //
Schiff
                }
           }
           return true;
     }

     enum Feld {WASSER, SCHIFF, TREFFER};
     final int FELD_HOEHE = 6;
     final int FELD_BREITE = 10;
     final int[] SCHIFFS_GROESSEN = {4, 3, 3, 2, 1};
     /**
      * Erzeuge ein Spielfeld der Groesse 10x6 mit mehreren Schiffen
unterschiedlicher Groesse
      * return: Das Spielfeld
      */
     Feld[][] setup() {
          Feld[][] feld = erzeugeSpielfeld(FELD_HOEHE,
FELD_BREITE);

           // erzeuge Schiffe (der jeweiligen Groesse)
           for (int i = 0; i < SCHIFFS_GROESSEN.length; i++) {
                int schiffsBreite, schiffsHoehe;
                if (Math.random() < 0.7) {
                      // 70 % Schiffs-Ausrichtung in x-Richtung
                      schiffsBreite = SCHIFFS_GROESSEN[i];
                      schiffsHoehe = 1;
                } else {
                      // Schiffs-Ausrichtung in y-Richtung
                      schiffsBreite = 1;
                      schiffsHoehe = SCHIFFS_GROESSEN[i];

                }

                int anzahlVersuche = 0;
                boolean erfolgreich;
                do {
                      int startPosX = (int) (Math.random() *
(FELD_BREITE - schiffsBreite + 1));
                      int startPosY = (int) (Math.random() *
(FELD_HOEHE - schiffsHoehe + 1));
                      anzahlVersuche++;
                      erfolgreich = setSchiff(feld, startPosX,
startPosY, schiffsBreite, schiffsHoehe);

//                    System.out.println("Erzeuge Schiff Nr. " + i +
": schiffsBreite = " + schiffsBreite + ", schiffsHoehe = " +
schiffsHoehe
//                                + ", startPosX = " + startPosX + ",
startPosY = " + startPosY
//                                + ", erfolgreich = " + erfolgreich +
", anzahlVersuche = " + anzahlVersuche);
```

```
                } while ((! erfolgreich) && (anzahlVersuche < 100));

            }

            return feld;
        }

}
```

Zusatz-Aufgabe 6 (für Java-Experten): Java-Rätsel: Java-Typen

Falls Sie noch Zeit und Lust haben, können Sie auch noch die folgende Aufgabe bearbeiten. Die Java-Rätsel auf diesem und folgenden Aufgabenblättern sind für diejenigen unter Ihnen gedacht, welche gerne anspruchsvolle Aufgaben lösen wollen, mit denen Sie noch tiefer in die Feinheiten von Java einsteigen können.

Hinweise:

- Bei den Java-Rätsel Aufgaben sollten Sie zunächst „im Kopf“ versuchen, das Programm zu analysieren, ohne es am Rechner einzugeben.

  Anmerkung: Wenn Sie verstehen, was das Programm voraussichtlich macht, haben Sie schon einmal den „normalen“ Stoff verstanden. Bei den Rätselaufgaben gibt es dann teilweise noch Besonderheiten, die zu unerwarteten Ergebnisse führen.

- Erst danach sollten Sie untersuchen, was Java mit dem Programm macht: Was ist die Ausgabe des Programms? Oder gibt es gar einen Compiler-Fehler?
- Versuchen Sie dann zu ergründen, warum das Programm es sich (vermutlich) anders verhält, als Sie ursprünglich gedacht haben.
- Bei Fragen wenden Sie sich an den Dozenten (oder schauen Sie in der Beispiellösung nach).

a) Rätsel (Long-Dision): Was gibt das folgende Programm aus – und warum?

```
public class LongDivision {
    public static void main(String[] args) {
        final long MICROS_PER_DAY = 24 * 60 * 60 * 1000 * 1000;
        final long MILLIS_PER_DAY = 24 * 60 * 60 * 1000;
        System.out.println(MICROS_PER_DAY / MILLIS_PER_DAY);
    }
}
```

Erwartete Ausgabe:

1000

- Wir definieren 2 Konstanten (final → Variable kann nicht mehr geändert werden → Konstante) `MICROS_PER_DAY`[1] und `MILLIS_PER_DAY`, welche die Anzahl der Microsekunden und der Millisekunden pro Tag enthalten.
- Anschließend teilen wir beide, sodass als Ergebnis genau **1000** (= Anzahl der Microsekunden pro Sekunde) übrig bleiben sollte.

Tatsächliche Ausgabe:

5

Erklärung:

- Warum rechnet Java den so falsch? Immerhin haben wir extra die Variablen (Konstanten) als long definiert, damit die Werte da auf jeden Fall hineinpassen:
  - `MICROS_PER_DAY: 86400000000`
  - `MILLIS_PER_DAY: 86400000`
- Das Problem ist jedoch, dass zwar die Variablen, d.h. der Speicherplatz für das Ergebnis groß genug ist (`long`), die Rechnung selbst jedoch auf `int`- Zahlen durchgeführt wird:
  - Die Zahlen 24, 60 und 1000 sind standardmäßig `int` in Java
  - Folglich erfolgt die Multiplikation von `int`-Zahlen im `int`-Zahlenbereich.
  - Dabei gibt es nun bei `MICROS_PER_DAY` einen **Zahlbereichs-überlauf**, da 86400000000 nicht mehr als `int` mit 32-Bit darstellbar ist (`Integer.MAX_VALUE` = 2147483647 = 2^{31}-1)
  - → wir erhalten als falschen Wert der Multiplikation `500654080`, der dann bei der Zuweisung an `MICROS_PER_DAY` in ein `long` umgewandelt wird (da ist jedoch bereits zu spät!)
  - → für die Division erhalten wir damit als Folgefehler 5 (statt 1000)
- **Lösung**:
  - Wir müssen mit long-Zahlen (statt mit int-Zahlen) rechnen
  - Dies erreichen wir, indem alle (oder zumindest die erste Zahl) nach long konvertieren:
    1. Entweder per Cast: ((long) 24)
    2. Oder (einfacher) **per Suffix L** (für long): **24L**
- **Fazit**:
  - Die Auswertung einer Zuweisung **`variable = wert`** (z.B. MICROS_PER_DAY = 24 * 60 * …) erfolgt in 2 Schritten:

[1] Hinweis: Per Konvention schreibt man in Java bei Konstanten alle Zeichen in Großbuchstaben.

1. Zunächst erfolgt die Berechnung des Wertes des Ausdrucks **`wert.`**
   Dies erfolgt stets mit den Typen, die in dem Ausdruck selbst vorkommen.
   (Hier: Rechnung als `int` mit Zahlüberlauf)
2. Erst nachdem das Ergebnis berechnet wurde (von einem Ausdruck oder einem Methodenaufruf), wird das Ergebnis in den gewünschten Typ der Ziel-Variable umgewandelt.
   (Hier: Automatisches Casting des `int`-Wertes nach `long`).

- **Achtung, gefährliche Falle**: Ein Cast vor dem Ausdruck wandelt ebenfalls nur das fertige Ergebnis in den Zieltyp um, ändert jedoch nichts an der Rechnung, d.h.
  - long MICROS_PER_DAY = **(long)** 24 * 60 * …
    → **bleibt fehlerhaft** (Rechnung erfolgt als `int`, da alle Zahlen vom Typ `int` sind), **erst Ergbnis wird gecastet**
  - Korrekt: long MICROS_PER_DAY = **((long) 24)** * 60 * …
    → Zahl 24 wird von `int` nach `long` gecastet. Folglich erfolgt Rechnung als `long`
    → hier müssen Klammern um die Zahl gesetzt werden (Multiplikations-Rechnung hat höhere Priorität als der Cast-Operator)
  - Korrekt (Alternative): long MICROS_PER_DAY = **24L** * 60 * …
    → Zahl `24L` ist eine `long`-Zahl (kein `int`), folglich wird ebenfalls im `long`-Zahlenbereich gerechnet
- Analog geschieht dies bei Methodenaufrufen fun(wert);

→ Korrigiertes Programm:

```
public class LongDivision_Lsg {
    public static void main(String[] args) {
        final long MICROS_PER_DAY_INT
                        = 24 * 60 * 60 * 1000 * 1000;
        final long MICROS_PER_DAY_LONG
             = 24L * 60 * 60 * 1000 * 1000;// 1x long reicht
//      final long MICROS_PER_DAY_LONG
//           = 24L * 60L * 60L * 1000L * 1000L;
        final long MILLIS_PER_DAY = 24 * 60 * 60 * 1000;

        System.out.println("Rechnung als int (fehlerhaft): "
                        + MICROS_PER_DAY_INT / MILLIS_PER_DAY);
        System.out.println("Rechnung als long (korrekt): "
                        + MICROS_PER_DAY_LONG / MILLIS_PER_DAY);

        System.out.println("Integer.MAX_VALUE = "
                        + Integer.MAX_VALUE);
```

```
            System.out.println("MICROS_PER_DAY_INT = "
                               + MICROS_PER_DAY_INT);
            System.out.println("MICROS_PER_DAY_LONG = "
                               + MICROS_PER_DAY_LONG);
            System.out.println("MILLIS_PER_DAY = "
                               + MILLIS_PER_DAY);
        }
    }
```

→ Ausgabe (Korrigiertes Programm):

```
Rechnung als int (fehlerhaft): 5
Rechnung als long (korrekt): 1000
Integer.MAX_VALUE = 2147483647
MICROS_PER_DAY_INT = 500654080
MICROS_PER_DAY_LONG = 86400000000
MILLIS_PER_DAY = 86400000
```

---

b) Rätsel (Casting): Was gibt das folgende Programm aus – und warum?

```
public class Multicast {
    public static void main(String[] args) {
        byte b = (byte) -1;
        char c = (char) b;
        int i = (int) c;
        System.out.println("i = " + i);
    }
}
```

Erwartete Ausgabe:

-1

- Wir casten die Zahl -1 jeweils in verschiedene Zahlbereiche, aber da sollte doch jeweils die gleiche Zahl bei herauskommen (-1 ist klein genug, dass da keine Informationen verloren gehen sollten).
- Konkret:
  - Sowohl `byte` (8-Bit) als auch `char` (16-Bit in Java, da Unicode) sind Untertypen von `int` (32-Bit)
  - -1 passt auf jeden Fall in `byte` hinein (`byte` = -128 … +127),
  - Damit sollte die Umwandlung `byte` → `char` → `int` unkritisch sein (da das jeweils Untertypen sind)
- Also: Ergebnis -1 erwartet!

Tatsächliche Ausgabe:

65535

Erklärung:

- Obige Erklärung ist fast korrekt, hat jedoch eine Kleinigkeit übersehen: die Zahlen (u.a. `int` und `byte`) sind vorzeichenbehaftete Typen in Java, während `char` (Zeichen) vorzeichenlos ist.
- Folglich geschieht hier das Folgende:
  - `byte b = (byte) -1;`

    Die Zahl -1 ist eine int-Zahl (siehe Rätsel LongDivision zuvor). Binär dargestellt (mit 32-Bits) ergibt sich[2]: $(11111111\ 11111111\ 11111111\ 11111111)_2 = (-1)_{10}$

    Bei der Konvertierung in `byte` (8-Bit) werden einfach die obersten 3 Byte „weggeworfen“, übrig bleibt das unterste Byte, welches weiterhin die Zahl -1 repräsentiert (da -1 als `byte` darstellbar ist, gibt es hier keinen Informationsverlust – es werden lediglich einige „Vorzeichenbits“ gelöscht): $(1111\ 1111)_2 = (-1)_{10}$
  - `char c = (char) b;`

    Bei der Umwandlung von `byte` (8-Bit) nach `char` (16-Bit) wird umgekehrt wie oben vorgegangen: Es wird das höchste Bit der (kleinen) Zahl, d.h. das Vorzeichenbit, dupliziert.
    Da die Zahl -1 negativ ist, ist das höchste Bit eine 1 (d.h. die Zahl ist negativ), und wir erhalten die 16-Bit Zahl
    $(1111\ 1111\ 1111\ 1111)_2 = (-1)_{10}$.

    Dieser Vorgang nennt sich Vorzeichenerweiterung („sign extension“) und wird bei allen Umwandlungen von vorzeichenbehafteten (ganzen) Zahlen ausgeführt:
    - Duplizierung des Vorzeichenbits beim **„Vergrößern“ des Zahlenbereichs** → dabei bleibt die Zahl erhalten,
    - Weglassen der höherwertigen (Vorzeichen-)Bits beim **„Verkleinern“ des Zahlenbereichs** (z.B. `int` → `byte`) → dabei kann die Zahl sich jedoch ändern, wenn die höherwertigen Bits nicht nur Vorzeichenbits sind sondern tatsächlich Daten enthalten, d.h. die Zahl nicht im kleineren Zahlenbereich darstellbar ist..

    Da jedoch `char` vorzeichenlos ist, wird die entstehende Zahl als positive ganze Zahl interpretiert, d.h.

[2] Siehe Digitaltechnik aus dem ersten Semester: Sie erhalten die negierte Darstellung $-x$ einer Binärzahl x durch Bildung des Zweierkomplements: Alle Bits von x werden invertiert, und anschließend wird 1 addiert, d.h. $-x = \overline{x} + 1$. Dasselbe Ergebnis ergibt sich bei n-Bits durch die Rechnung $-x = 2^n - x$, d.h. bei 8-Bits haben wir $-1 = 2^8 - 1 = 255 = (1111\ 1111)_2$.

$(1111\ 1111\ 1111\ 1111)_2 = (65.535)_{10}$ (als `char`)

- `int i = (int) c;`

  Da `char`-Zeichen kein Vorzeichen haben (d.h. unsigned Werte sind), wird hier beim Vergrößern des Zahlenbereichs nicht das höherwertigste (Vorzeichen)-Bit dupliziert, sondern die Zahl mit Nullen aufgefüllt:

  $(11111111\ 11111111)_2$ (als `char`)
  → (**00000000 00000000** $11111111\ 11111111)_2$ (als `int`)

  Letztere Zahl entspricht nun genau **65.535**, die unser Programm auch ausgibt.

- **Lösung:**
  - Wenn wir stattdessen eine vorzeichenbehaftete Erweiterung haben wollen, müssten wir an dieser Stelle explizit den 16-Bit vorzeichenlosen `char`-Wert in einen – bitweise identischen – 16-Bit vorzeichenbehafteten `short`-Wert umwandeln.
  - Dieser wird dann – wie erwartet – bei Umwandlung nach 32-Bit `int` mittels **Vorzeichenerweiterung („sign extension“)** wie folgt umgewandelt, wobei die -1 erhalten bleibt:

    $(11111111\ 11111111)_2 = (65.535)_{10}$ (als `char`)
    → $(11111111\ 11111111)_2 = (-1)_{10}$ (als `short`)
    → (**11111111 11111111** $11111111\ 11111111)_2 = (-1)_{10}$ (als int)

- **Fazit:**
  - Vermeide die Mischung von `char`-Werten mit ganzen Zahlen: `char` repräsentiert Zeichen, während `byte/short/int/long` ganze Zahlen (mit Vorzeichen) repräsentiert.
  - Lediglich wenn Sie explizit einen ASCII-Code oder Unicode in das Zeichen umwandeln wollen, macht eine Konvertierung überhaupt Sinn. Aber auch dafür sollte man besser lieber die bereits vordefinierten Methoden der Klasse **`Character`** verwenden, um auch Sonderfälle wie z.B. einige Unicode-Zeichen, die durch 2 `char`-Zeichen repräsentiert werden, korrekt abzubilden (siehe http://docs.oracle.com/javase/6/docs/api/index.html?java/lang/Character.html).

→ Erweitertes Programm (Mit Ausgabe der Zwischenwerte & Vorzeichenerweiterung):

```
public class Multicast_Lsg {
   public static void main(String[] args) {
      byte b = (byte) -1;
      char c = (char) b;
      int i = (int) c;
      System.out.println("b = " + b);
      System.out.println("c = " + c);
```

```
        System.out.println("i = " + i);

        // Umwandlung in einer Zeile
        System.out.println((int) (char) (byte) -1);

        // Zahlumwandlung mit Sign-Extension bei char:
        int i2 = (short) c; // Cast bewirkt Vorzeichenerweiterung
        System.out.println("i2 = " + i2);
    }
}
```

→ Ausgabe (Erweitertes Programm):

```
b = -1
c = ?
i = 65535
65535
i2 = -1
```