**Schiffe versenken – Bauplan – Version „EINFACH“**

**Ziel**: Du kannst dein Wissen über Klassen, Objekte, Methoden, if-Abfragen, Arrays, Scanner und Schleifen sinnvoll anwenden.

Falls du das Spiel nicht kennst, dann kannst du z.B. hier nachlesen, wie es gespielt wird: <https://de.wikipedia.org/wiki/Schiffe_versenken>

In blauer Farbe sind die Teile markiert, die ihr in euerem Code schon haben solltet.

**Ein Bild, das Text, Diagramm, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung**

Klasse „*Ship*“

Ein Schiff ist Teil des Spiels und soll später auf dem Spielfeld platziert werden können. Vorerst legst du als Programmierer fest, wie viele Schiffe auf dem Spielfeld platziert werden dürfen und welche Länge diese haben. Es gibt immer unterschiedlich lange Schiffe.

* Erstelle eine Klasse „*Ship*“
* Du benötigst die Attribute
  + lenght: Für die Länge des Schiffes
  + hitCount: Für einen Zähler, wie oft ein Schiff getroffen wurde
  + isSunk: Ob das Schiff versenkt wurde oder nicht
* Da wir verschiedene lange Schiffe erzeugen möchten, benötigen wir einen anderen Konstruktor, als den Default-Konstruktor.
  + Unser Konstruktor muss eine Länge übergeben bekommen, die die Länge des Schiffes festsetzt.
  + Der hitCount muss zu Beginn 0 sein
  + Da das Schiff zu Beginn noch schwimmt, muss isSunk false sein
* Dann brauchen wir für das Schiff noch Methoden:
  + getLenght(): Die Methode soll die Länge des Schiffes zurückgeben
  + getHitCount(): Die Methode soll den hitCount zurückgeben
  + isSunk(): Die Methode soll isSunk zurückgeben
  + hit(): Diese Methode soll
    - falls ein Schiff nicht gesunken ist bzw. isSunk weiterhin false (prüfe: !isSunk) den hitCount um einen erhöhen
      * falls dann der hitCount == die Länge vom Schiff ist, dann soll isSunk auf true gesetzt werden und eine Meldung an den Spieler gegeben werden, dass das Schiff gesunken ist

Klasse „*Board*“

Zum Spielen benötigen wir auch ein Spielfeld. Das ist unser Board. Auf dem Board können die Schiffe platziert werden.

* Erstelle eine Klasse „*Board*“
* Du benötigst die Attribute
  + grid: Ein zweidimensionales Array, das das Spielfeld darstellt. Jeder ~ ist Wasser, S steht für ein Schiff.
  + ships: Ein Array, das die Schiffe des Spiels speichert
  + size: Die Größe des Spielfelds (z.B. 10x10)
* Wenn wir ein Objekt des Boards erstellen, soll auch direkt ein passender Array erstellt werden und initial mit Wasser gefüllt werden
  + Erstelle einen Konstruktor, der eine beliebige Größe für das Spielfeld übergeben bekommt und instanziiere damit size, das grid und ships
  + Als Programmierer entscheidest du, dass das Array von ships die Größe 2 haben soll
  + Jetzt müssen wir noch unser grid mit Wasser ~ befüllen
    - Schreibe dazu zwei verschachtelte for-Schleifen, die über das ganze Array grid iterieren
    - Bei jedem Iterationsschritt soll der Eintrag grid[i][j] = ‚~‘ gesetzt werden
* Da das Board verschiedene Dinge ermöglichen muss, benötigen wir hierfür auch Methoden
  + Wir müssen ein Schiff auf dem Spielfeld platzieren können, dafür benötigen wir eine Methode, die ein Schiff, was auf dem Feld platziert werden soll, übergeben bekommt und eine x- und y-Koordinate auf dem Spielfeld. Zudem benötigen wir noch die Information, ob das Schiff horizontal oder vertikal platziert werden soll. Hierfür nutzen wir einen boolean. placeShip(Ship ship, int x, int y, boolean horizontal) Unsere Annahme ist, dass das Schiff von den Startkoordinaten entweder nach unten oder rechts platziert wird.
    - Dann muss die Methode zuerst überprüfen, ob die versuchte Platzierung noch auf dem Spielfeld liegt. Wir können das nicht später machen, weil, wenn das Schiff nicht auf dem Spielfeld liegt, jeder Platzierungsversuch mit einer Fehlermeldung beendet wäre.
      * Unterscheide zunächst in einer Abfrage, ob das Schiff horizontal oder vertikal platziert werden soll.
        + Falls das Schiff horizontal platziert werden soll, dann muss mit einer weiteren Bedingung geprüft werden, ob die x-Koordinate (die x-Achse ist die horizontale Ebene) plus die Länge vom Schiff, größer sind als die Größe vom grid. Ist das so, soll die Methode beendet werden (return) und vorher der Nutzer informiert werden
        + Falls das Schiff vertikal platziert werden soll, dann muss mit einer weiteren Bedingung geprüft werden, ob die y-Koordinate (die y-Achse ist die horizontale Ebene) plus die Länge vom Schiff, größer sind als die Größe vom grid. Ist das so, soll die Methode beendet werden (return) und vorher der Nutzer informiert werden
      * Jetzt können natürlich auch noch andere Schiffe im Weg liegen und verhindern, dass wir unser Schiff platzieren können. Um das zu kontrollieren iterieren wir mit einer Schleife über die potentiellen Felder für das zu platzierende Schiff
        + Für die zu überprüfenden Koordinaten müssen wir wieder zwischen horizontal und vertikal unterscheiden

Falls wir horizontal platzieren wollen, dann bleibt die y-Koordinate unverändert und die x-Koordinate wird durch die Schleifeniteration schrittweise so erhöht, dass jedes nachfolgende Feld einmal geprüft werden kann

Falls wir vertikal platzieren wollen, dann bleibt die x-Koordinate unverändert und die y-Koordinate wird durch die Schleifeniteration schrittweise so erhöht, dass jedes nachfolgende Feld einmal geprüft werden kann

Die so für jeden Iterationsschritt erhaltene potenzielle Position, muss überprüft werden, ob hier Wasser ist oder nicht. Wenn Kein Wasser an dieser Stelle ist, dann soll der Spieler eine Meldung bekommen und die Methode soll beendet werden

* + - * Jetzt haben wir alle möglichen Fehlerquellen geprüft. Sind wir bis hierher gekommen, müssen wir natürlich das Schiff noch platzieren. Dafür iterieren wir wieder mit einer Schleife über die potentiellen Felder für das zu platzierendes Schiff.
        + Für die Festlegung der Koordinaten müssen wir wieder zwischen horizontal und vertikal unterscheiden

Falls wir horizontal platzieren wollen, dann bleibt die y-Koordinate unverändert und die x-Koordinate wird durch die Schleifeniteration schrittweise so erhöht, dass jedes nachfolgende Feld einmal geprüft werden kann

Falls wir vertikal platzieren wollen, dann bleibt die x-Koordinate unverändert und die y-Koordinate wird durch die Schleifeniteration schrittweise so erhöht, dass jedes nachfolgende Feld einmal geprüft werden kann

* + - * + Die so für jeden Iterationsschritt erhaltene Position, muss nun als Schiff ‚S‘ gesetzt werden im grid
    - Wenn du jetzt noch möchtest, kannst du am Ende deiner Methode den Spieler das Feedback geben, dass das Schiff platziert wurde
  + Die Hauptattraktion des Spiels ist das Abschießen von Schiffen. Damit benötigen wir noch eine Methode, die das auch auf dem Spielfeld umsetzen kann. Damit die Methode weiß, an welcher Position ein Schuss abgegeben wurde, werden die x- und y-Koordinaten dieser Position benötigt. Bsp. shootAt(int x, int y)
    - Zu Beginn muss sichergestellt werden, dass an der Position auf dem grid kein Wasser oder aber auch Treffer ist.
      * Ist das nicht der Fall, dann können wir über unsere vorhandenen Schiffe iterieren
        + Finden wir dann ein Schiff auf dem Feld und auf dem grid ist auch noch an der Position ein Schiff vermerkt, dann können wir dem Schiff einen Treffer vermerken. ship.hit()
        + Dann sollten wir auch gleichzeitig auf dem grid an der Position einen Treffer vermerken
      * Für die Implementierung der weiteren Logik ist es wichtig, dass die Methode zurückmeldet, dass sie einen Treffer oder einen Fehlschuss hatte. In diesem Fall ist es ein Treffer, damit soll die Methode als Rückgabewert ein true zurückgeben
    - Andernfalls muss das Feld mit einem Fehlschuss markiert werden
      * Für die Implementierung der weiteren Logik ist es wichtig, dass die Methode zurückmeldet, dass sie einen Treffer oder einen Fehlschuss hatte. In diesem Fall ist es ein Treffer, damit soll die Methode als Rückgabewert ein true zurückgeben
  + Damit das Board weiß mit welchen Schiffen es arbeitet, muss das Board eine Methode zur Verfügung stellen, die das Array ships mit allen im Spiel befindlichen Schiffen befüllt
    - Wir müssen dafür in dem Array ships mit Hilfe einer Iteration über das Array eine freie Stelle finden und es dort hinzufügen. Ist es platziert, so soll die Schleife halten. Das kannst du mit dem Schlüsselwort break; erreichen
  + Damit der Spieler auch visuell sehen kann, wie der Spielstand ist und sich das Spielfeld nicht merken muss, braucht das Board auch noch eine Funktion, um das Spielbrett in der Konsole auszugeben.
    - Itertiere dafür mit zwei verschachtelten for-Schleifen über das grid und gebe jede einzelne Zelle aus
  + Damit wir von anderer Stelle auf unsere ships und die size zugreifen können, benötigen wir hier auch noch zwei getter-Methoden, die die ships und die size zurück geben

Klasse „*BattleshipGame*“

Das ist das Herzstück des Spiels. Hier bauen wir das Spiel zusammen. Damit das klappt benötigen wir hier noch ein paar Schritte.

* Du benötigst folgenden Attribute
  + enemyBoard: Ein gegnerisches Spielfeld, von dem du als Spieler die Schiffe erraten möchtest – nur die Deklaration
  + scanner: Da du den Scanner für die Interaktion häufig benötigst, kannst du ihn auch vorher deklarieren
* Dann benötigen wir einen Konstruktor von BattleshipGame
  + Hier möchten wir jetzt den Spieler fragen, wie groß das Spielfeld sein soll
    - Die Antwort speichern wir und instanziieren darauf ein neues Board-Objekt mit der gespeicherten Größe
  + Jetzt benötigen wir noch Schiffe für das Spielfeld
    - Instanziiere zwei Schiffe unterschiedlicher Längen
  + Füge über unsere addShip()-Methode vom Board beide Schiffe dem Board hinzu. enemyBoard.addShip(enemyShip1)
  + Plaziere mit Hilfe der placeShip-Methode beide Schiffe auf dem Board. Bsp. enemyBoard.placeShip(enemyShip1, 1, 1, true)
* Dann benötigen wir noch jeweils eine Methode zum Starten und Enden des Spiels, sowie ein Main-Methode, die Spielausführung startet.
  + Beginnen wir mit der start-Methode
    - Das Spiel soll so lange laufen, bis es gewonnen oder verloren wurde. Dafür definieren wir uns einen boolean, der erstmal false ist. Das steht dafür, dass das Spiel noch nicht gewonnen oder verloren ist.
      * Solange der boolean false ist (while(!gameOver))
        + Soll einmal das gegnerische Feld ausgegeben werden
        + Eine Abfrage an den Spieler gehen, welche Koordinaten er beschießen möchte
        + Die Eingabekoordinaten werden in Variablen gespeichert
        + Wenn du dich erinnern kannst, hat das Board eine Methode zum Schießen. Diese Methode gibt uns einen boolean zurück, ob ein Treffer oder kein Treffer erfolgte.

Falls der Methodenaufruf true zurückgibt, dann soll dem Spieler mitgeteilt werden, dass er getroffen hat

Andernfalls soll ihm der Fehlschuss mitgeteilt werden

* + - * Zum Schluss kommt der Methodenaufruf, ob das Spiel zu Ende ist. Wenn das der Fall ist, soll der boolean gameOver auf true gesetzt werden
  + Die Methode zur Überprüfung, ob das Spiel zu Ende ist, prüft mit Hilfe einer Iteration über die in ships auf dem Board enthaltenen Schiffe, ob alle Schiffe gesunken sind (isSunk ist true).
    - Ist das der Fall, soll die Methode true zurückgeben
    - Ist das nicht der Fall, soll die Methode false zurückgeben
  + Zum Schluss benötigen wir noch eine Methode, die das Ganze startet. Hierfür nutzen wir eine Main-Methode.
    - Wir erzeugen zunächst ein Instanziiertes Objekt unserer BattleshipGame-Klasse
    - Dann rufen wir auf dem Objekt die start()-Methode auf

Wenn du alles Implementiert hast, solltest du auf ganz simpler Weise gegen den Computer Schiffe versenken spielen können. 😊 – Denke an JavaDoc!

**Schiffe versenken – Bauplan – Version „FORTGESCHRITTEN“**

Voraussetzung ist, dass du eine einfache Implementierung des Schiffes versenken hast!

**Ziel**: Du kannst Code vereinfachen und einfache Konzepte der Vererbung berücksichtigen.

* Die Methode zum Platzieren von Schiffen ist sehr ausführlich und sehr kleinschrittig. Versuche die Methode so umzuschreiben, dass sie möglichst wenig Wiederholungen hat.
* Es ist sehr simpel nur ein universelles Schiff zu verwenden
  + Erstelle mindestens zwei Arten von Schiffen, die von der Klasse Ship erben
  + Erzeuge die Objekte dieser Schiffe in der Klasse BattleshipGame, anstatt normaler Ships
  + Schreibe dein Konstruktor von Ship so um, dass du keine Länge mehr übergeben musst. Diese soll jetzt bei den Schiffstypen fest hinterlegt sein und nicht änderbar (Stichwort: final)
  + Ändere in der Platzierung von Schiffen auf dem Board den Buchstaben so, dass du für jeden unterschiedlichen Schiffstypen einen anderen Buchstaben vergeben kannst.

**Schiffe versenken – Bauplan – Version „KÖNNER I“**

Voraussetzung ist, dass du eine fortgeschrittene Implementierung des Schiffes versenken hast!

**Ziel**: Du kannst eigenständig Konzepte entwickeln und integrieren. Dafür kannst du fehlende Wissenslücken eigenständig schließen.

* Schreibe deine Methode start() im BattleshipGame so um, dass der Computer die Schiffe automatisch platziert

**Schiffe versenken – Bauplan – Version „KÖNNER II“**

Voraussetzung ist, dass du eine Könner I Implementierung des Schiffes versenken hast!

**Ziel**: Du kannst eigenständig komplexere Konzepte entwickeln und integrieren. Dafür kannst du fehlende Wissenslücken eigenständig schließen.

* Ergänze das Schiffe versenken so, dass du selbst ein Spielfeld mit Schiffen bestücken kannst.
* Schreibe eine kleine Klasse, die automatisch Koordinaten-Votes abgibt, die du auf dein eigenes Feld anwenden lässt. -> Baue dir also einen Computer, der gegen dich spielt. Wechselt eure Züge ab.