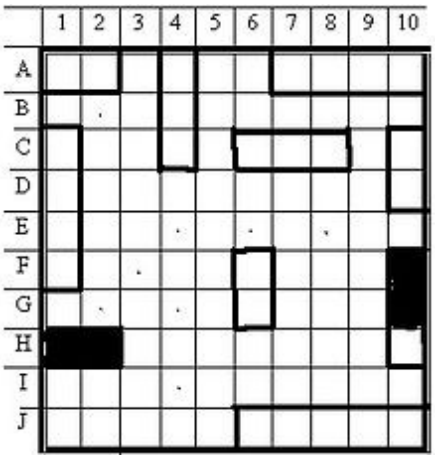
Aufgabe 06: Design Pattern - Schiffe Versenken

Die 1-Spieler Version des Spiels „Schiffe versenken“ soll implementiert werden. Der Computer berechnet zufällig die Anordnung der Flotte auf z.B. einem *10x10* Spielfeld (die Größe ist Variabel). Anders als in der bekannten Version versucht hier nur der Spieler alle Schiffe der Flotte des Computers zu versenken.

<u>Folgende Regeln gelten für das Setzen der einzelnen Schiffe:</u>



- ✓ Die Schiffe dürfen nicht aneinander stoßen und sich nicht berühren.
- ✓ Die Schiffe dürfen nicht über Eck gebaut sein oder Ausbuchtungen besitzen.
- ✓ Die Schiffe dürfen auch direkt am Rand liegen.
- ✓ Die Schiffe dürfen nicht diagonal aufgestellt werden.
- ✓ Jeder verfügt über insgesamt zehn Schiffe:
  - ein Schlachtschiff (5 Kästchen)
  - zwei Kreuzer (je 4 Kästchen)
  - drei Zerstörer (je 3 Kästchen)
  - vier U-Boote (je 2 Kästchen)

Feedback für den Spieler:

- ✓ Jeder Schuss ins Wasser wird mitgezählt und dem Spieler auf der Karte angezeigt.
- ✓ Wenn ein ganzes Schiff versenkt wurde, gibt der Computer darüber Bescheid.
- ✓ Wenn alle Schiffe versenkt wurden, ist das Spiel beendet.
- ✓ Man bekommt natürlich nur die Treffer angezeigt, die man tatsächlich schon geschafft hat (zusammen mit den Schüssen ins Wasser).

<u>Aufgabe 1.1</u>: Erstellen eines grundlegenden Klassendiagramms für das Spiel (das Spiel ist eine Anwendung mit grafischer Benutzeroberfläche - diese muss aber nur angedeutet und nicht im Detail im Klassendiagramm abgebildet werden, entscheidend ist nur die Rolle der *GUI* als *Observer*). Auch Konstruktoren und Getter- bzw. Setter-Methoden müssen nicht hinein.

Bitte dabei folgendes beachten:

- ✓ Es muss darauf geachtet werden, dass Darstellung, die Benutzereingaben und die Spiellogik klar voneinander getrennt sind (Stichwort: *MVC*-Pattern).
- ✓ Verwende für die Benachrichtigung der *GUI* den *Observer/Observable*-Pattern.
- ✓ Die *GUI*-Klasse muss nur grundsätzlich modelliert werden. Es müssen keine verwendeten Steuerelemente bzw. *GUI*-spezifische Methoden angegeben werden.

Die *Model*-Klassen müssen folgendes beinhalten:

- ✓ Die Information über die **Schiffe.** Ob diese in einer eigenen Klasse gespeichert werden oder ob die Abbildung direkt am Spieltfeld passiert ist freigestellt. Folgendes muss aber dabei berücksichtigt werden bzw. enthalten sein:
  - Es muss ersichtlich sein welche Teile zu welchem Schiff gehören, damit festgestellt werden kann, ob ein Schiff versenkt ist.
  - Somit ist auch die Länge der einzelnen Schiffe ermittelbar.
  - Eine Methode, die feststellt (und im positivem Fall speichert), ob ein Teil des Schiffes getroffen wurde. Dafür ist eine geeignete Datenstruktur zu finden.

- ✓ Eine **Spielfeld**-Klasse sollte existieren. Diese beinhaltet:

- Die einzelnen Schiffe bzw. deren Teile.
- Eine textuelle Ausgabe des Spielfeldes.
- Eine Überprüfung, ob das Spiel fertig ist, also alle Schiffe zerstört sind.
- Das Abgeben der Schüsse (die dann an die Schiffe bzw. deren Teile weitergegeben werden).

✓ Eine **Schiffe-Factory**, die für folgendes zuständig ist:
- Erzeugen der einzelnen Schiffe bzw. deren Teile. Eine statische Methode *create* bekommt die Länge des Schiffes und das aktuelle Spielfeld übergeben. Die Orientierung und die Position wird zufällig gewählt (Bsp.: *int z = (int)(Math.random()*10)*). Die Methode gibt ein Schiff-Objekt bzw. deren Teile zurück.
- Eine statische Methode *boolean schiffMöglich()* soll feststellen, ob sich ein Schiff mit den bereits bestehenden Schiffen bzw. deren Teilen überlappt. Die Parameter für diese Methode sind selbst zu finden.

Es kann jeder beliebige Ansatz gewählt werden, nur muss dieser stimmig sein und die Basis für die Aufgaben 1.2 bis 1.5. darstellen. Redundanzen in der Speicherung der Daten gilt es zu vermeiden!

Aufgabe 1.2: Bitte die Methode der *Spielfeld*-Klasse, die die Ausgabe des Spielfeldes übernimmt (am Besten unter Verwendung von *toString()*), implementieren. Ein Schiffteil soll als ‚*X*' dargestellt werden; wenn es getroffen ist, als ‚*O*'. Die Treffer, die ins Wasser gemacht wurden, sollen durch ein ‚W' markiert werden, alle verbleibenden Zellen werden durch "_" dargestellt.

Aufgabe 1.3: Bitte die Methode, die Überprüft, ob ein Schiff versenkt ist, implementieren.

Aufgabe 1.4: Weiters die Methode *schiffMöglich()* der *Factory*-Klasse implementieren. Hier zwei unabhängige Vorschläge für die Vorgangsweise:

1. Es werden alle 8 Nachbarpunkte eines Schiffteils einzeln kontrolliert – natürlich unter der Verwendung einer eigenen Methode. Vorsicht bei den Rändern.
2. Es sollen alle Teile eines Schiffes mit denen des zweiten Schiffes bezüglich der Abstände (sowohl x als auch y) verglichen werden. Hilfreich könnte hier die Methode: *int Math.abs(int a)* sein, die den Absolutwert (Betrag) berechnet (so wird die Anzahl der notwendigen Vergleiche verkleinert).

Aufgabe 1.5: Für welchen Zweck wird der Command-Pattern normalerweise verwendet? Bitte veranschaulichen Sie das anhand dieses Spiels (auch wenn die Anwendung dort nur bedingt Sinn macht) - und zwar grundlegend in *UML* und Quellcode.