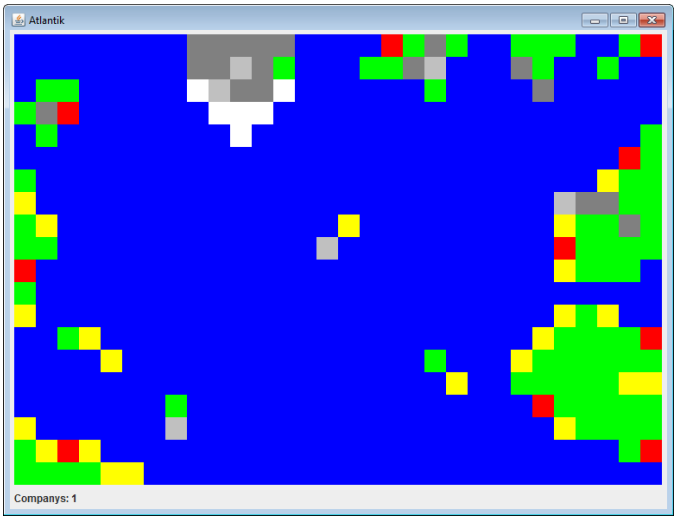
Der SeaTrade-Server beinhaltet eine Seekarte des Nordatlantik mit 10 Häfen, zwischen denen es Waren zu transportieren gilt.



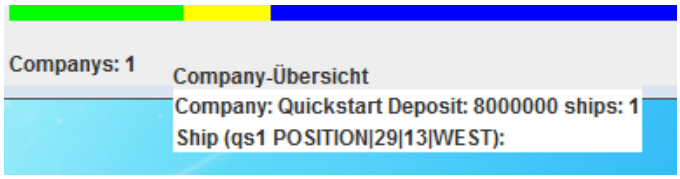
Rote Felder stellen die Häfen dar, blaue das mit Schiffen befahrbare Wasser.

Beim Klick auf einen Hafen werden Detailinformationen zu diesem Hafen dargestellt:

- **Name** des Hafens,
- die dort zum Transport bereitstehenden **Ladungen**
- und welche **Schiffe** sich derzeit im Hafen befinden.

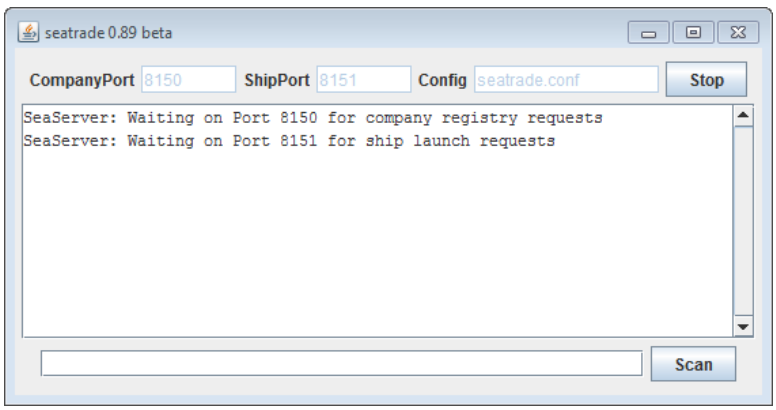
Jede Ladung (**Cargo**) hat dabei:
- Eine ID
- Starthafen
- Zielhafen
- Wert der Ladung

Ziel ist es eine Reederei (Company) zu gründen und durch den Transport von Ladungen Geld zu verdienen. Dazu kann die Reederei Schiffe erwerben, die dann im Auftrag der Reederei die Weltmeere befahren.



Ein Klick auf die untere Statuszeile zeigt Detailinformationen der **Companys**:
- Name,
- aktuelles Guthaben
- und alle Schiffe mit aktueller Position.

Im Admin-Fenster können die Portnummern für Companys bzw. Ships eingestellt werden.
In der Konfigurationsdatei „seatrade.conf" können globale Einstellungen angepasst werden. Die Bedeutung ist in der Datei beschrieben. Bei ‚Start' werden die Einstellungen eingelesen.
Im unteren Eingabefeld können Kommandos an den SeaTrade Server eingegeben werden:



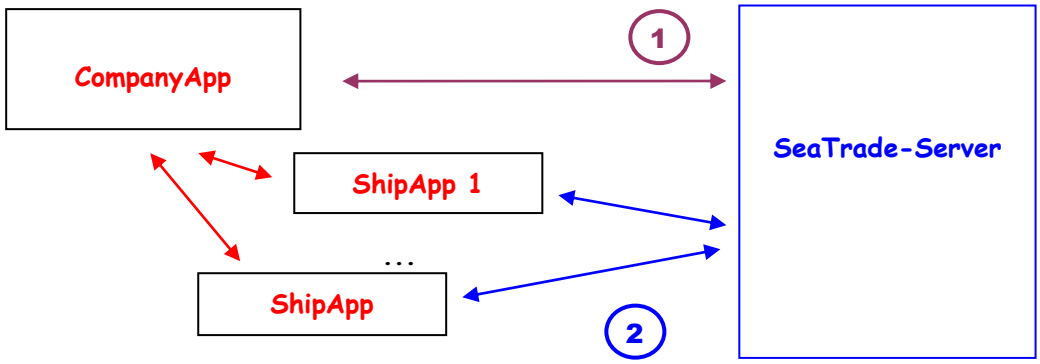
`companys`
gibt Infos zu allen Companys aus

`harbours`
gibt Infos zu allen Häfen aus

`cargo n`
Erzeugt an zufälligen Häfen insgesamt n neue Ladungen

Zu Testzwecken gibt es eine bereits aktive integrierte Default-Company „**`Quickstart`**", so dass eine ShipApp auch ohne eigene Company getestet werden kann.

Komponentenübersicht (**alle blauen/violetten Teile sind vorgegeben/implementiert**):



TODO (Basisanforderungen): Entwerfen und Implementieren sie …

1. … eine **CompanyApp**, die sich über das vorgegebene (Text-) Protokoll beim SeaTrade-Server registriert, Informationen holt und Aktualisierungen empfängt. ***Sie ist ein Client des SeaTrade-Servers und gleichzeitig ein Server für die ShipApp***(-Instanzen).
   Sie teilt ihren Schiffen neue Transportaufträge mit. (1)

2. … eine **ShipApp**, die sich über **ein selbst zu definierendes (Text-)Protokoll** beim Company-Server anmeldet, von dort Instruktionen erhält und sich über das **vorgegebene (Text-)Protokoll** mit dem SeaTrade-Server verbindet um Transportaufträge auszuführen. ***Sie bleibt mit ihrer CompanyApp in Verbindung und übermittelt angefallene Transportkosten und Erlöse durch zugestellte Ladungen***, so dass die CompanyApp immer den Überblick über das aktuelle Guthaben hat! (2)

3. Alle **Komponenten** (CompanyApp & ShipApps) **müssen auf unterschiedlichen Rechnern laufen** können. Alle Multi-Thread-Komponenten müssen „**threadsafe**" sein. Beim Abmelden/Sinken eines Schiffes müssen die jeweiligen Sessions aus der CompanyApp-Verwaltung entfernt werden.

TODO (Erweiterungen – zur Auswahl, je nach Neigung, je mehr desto besser):

4. Implementierung CompanyApp und/oder ShipApp als **GUI-Anwendung** (java.awt oder javax.swing)

5. **Unterstützung erweiterter Funktionalitäten des SeaTrade-Servers** , durch Implementierung von zusätzlichen Nachrichten des Textprotokolls. Vorschläge: (2)
   - Einfach zu bedienende, manuelle Schiffssteuerung innerhalb der ShippApp
   - Anforderung und Anzeige von Radarmessungen im Umfeld des Schiffs um Kollisionen mit Schiffen und unbefahren Gelände vermeiden zu können.
   - Eventuelle Anzeige der Grobrichtung zum nächten Zielhafen, da ja im Schiff keine gesamte Seekarte vorhanden ist und eine Radarmessung nur die umliegenden Felder liefert.
   - Erweiterte Ladungsaufnahme anhand der Cargo-ID, um gezielt bestimmte Ladungen aufzunehmen.

6. … (weitere Features nach Absprache)

„Lieferumfang" seatrade_0.97.zip:

`seatrade.jar` // ausführbares Java-Archiv des Servers
Package `sea` // Java-Quelltexte der beschriebenen Hilfsklassen/Enums

Im mitgelieferten `package sea` gibt es einen ganzen Satz fertiger Hilfsklassen (die unverändert bleiben!):

`enum Ground`: Bodenbeschaffenheit eines Feldes
`enum Direction`: Himmelsrichtung in die der ein Schiff aktuell zeigt
`class Size`: Speichert die Grösse (breite, hoehe) der Seekarte
`class Position`: Aktuelle x/y-Position eines Objekts auf der Seekarte (ggf. mit Direction)
`class Topography`: Topographie eines Feldes (Ground mit Höhenangabe)
`class Cargo`: Abbildung einer Ladung
`class RadarScreen, RadarFieldm RadarDirection`: Abbildung eines Radarscreens bestehend aus den einzelnen Feldern (RadarField) in der jeweiligen Richtung (RadarDirection) zum Mittelfeld

1 Beschreibung des Text-Protokolls zwischen CompanyApp und SeaTrade-Server:

Richtung	CompanyApp <-> SeaTrade	Beschreibung
C => S	register:companyname	Registrierung mit frei wählbarem Companyname beim Server
C <= S	registered:SIZE|width|height:deposit	Quittierung der Registrierung mit Seekartengröße und Startkapital oder
C <= S	error: text	Fehlerkennung
C => S	getinfo:harbour	Abfrage aller Häfen
C <= S	harbour:POSITION|x|y|NONE:name … endinfo	Liste aller Häfen mit Position und Hafenname … Abgeschlossen mit endinfo
C => S	getinfo:cargo	Abfrage alle momentan wartender Ladungen
C <= S	cargo:CARGO|id|src|dest|value … endinfo	Liste alle Ladungen mit Id, Start-/Zielhafen und Wert Abgeschlossen mit endinfo
S => C	newcargo: CARGO|id|src|dest|value	Asynchrone Aktualisierung, wenn neue Ladungen zum Verschicken eintreffen
C => S	exit	Company meldet sich ab

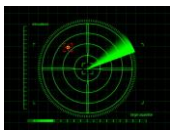
2 Beschreibung des Text-Protokolls zwischen ShipApp und SeaTrade-Server:

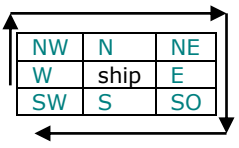
Richtung	ShipApp <-> SeaTrade	Beschreibung
C => S	launch:company:harbour:shipname	Stapellauf eines Schiffes der angegebenen Company(name) im angegebenen Hafen mit dem neuen Schiffsnamen
C <= S	launched:POSITION|x|y|dir:cost	Quittierung mit Schiffsposition und Baukosten
C <= S	error: text	oder Fehlerkennung
C => S	moveto:harbour	Auslaufen des Schiffs mit Angabe des Zielhafens
C <= S	moved:POSITION|x|y|dir:cost	Nach jeder Bewegung wird die neue Position gesendet
C <= S	reached:harbour	Nachricht wenn Zielhafen erreicht
C => S	loadcargo	Versucht Ladung aufzunehmen (geht nur im Hafen)
C <= S	loaded:CARGO|cargo_id|start|ziel|value	Quittung der Erfolgreicher Ladung
C <= S	error: text	oder Fehlerkennung
C => S	unloadcargo	Löscht Ladung
C <= S	unloaded:profit	Quittung mit erhaltenem Erlös
C <= S	error: text	oder Fehlerkennung
C => S	exit	Schiff meldet sich ab

Wenn in config-Datei **LEVEL>1** eingetragen ist, sind folgende **erweiterte** Befehle möglich:		
C => S	move:DIRECTION	Bewegt das Schiff manuell ein Feld in die angegebene Himmelsrichtung. Der „Steuermann" hat die volle Verantwortung! Bei ungültigen Zielfeldern, bzw. Schiffen auf dem Zielfeld kann das Schiff sinken.
C => S C <= S	radarrequest radarscreen:RSCREEN; POSITION|x|y|dir; RF|ground|S;...	Anforderung Radarmessung von der aktuellen Schiffsposition. Liefert einen Scan der umliegenden Felder (siehe RadarDirection,-Field,-Screen).
S => C	sunk:POSITION|x|y|dir	Asynchrone Nachricht, wenn Schiff gesunken ist
C => S	loadcargo**:id**	Versucht Ladung mit der angegebenen ID aufzunehmen (geht nur im Hafen, wenn dort eine Ladung mit dieser ID vorhanden ist). D.h. es muss nicht die erste Ladung zuerst eingeladen werden.

Wird der SeaTrade-Server mit LEVEL 1 (in Config-File einstellbar) gestartet, werden folgende Verhaltensweisen aktiviert:

- Generell gilt: Schiffe können kollidieren, ihre Ladung verlieren und eventuell sinken (insbesondere bei manuell gesteuerten Schiffen).
- Schiffe die automatisch (mit moveto:harbour) unterwegs sind, halten an und unterbrechen ihre Fahrt, sofern sie bei ihrem nächsten Zug auf ein Feld ziehen würden, auf dem schon ein Schiff steht. Allerdings laufen die Kosten weiter! Ist das Feld wieder frei, wird die Reise fortgesetzt.
- Sobald ein Schiff mit einem manuellen move-Befehl bewegt wird, muss die manuelle Steuerung mindestens bis zu einem Hafen fortgesetzt werden. Erst dann kann wieder der moveto-Befehl verwendet werden.
- In Häfen können nach wie vor beliebig viele Schiffe liegen.
- Cargos können gezielt unter Angabe der ID geladen werden, nicht nur die älteste Ladung.
- Eine Radarmessung liefert, ausgehend von der aktuellen Position des Schiffs, die Werte der 8 umgebenen Felder. Außerdem kann erkannt werden, ob sich auf einem Feld ein anderes Schiff befindet. Der geliefert Radarscreen ist unabhängig von der aktuellen Schiffsausrichtung, immer gleich in Nord-/Südrichtung ausgerichtet.





- Die RadarFelder sind unabhängig von der Schiffsausrichtung immer in der in RadarField angegebenen Reihenfolge abgelegt (West, NW, Nord, …, SW).

Erwartete Arbeitsergebnisse: E3FI2	**Bis zum:**
Entwurf des Grundsystems (mindestens der Basisanforderungen) bestehend aus UML-**Klassendiagramm**(en) und einer Beschreibung des geplanten **Kommunikationsprotokolls** zwischen Ship-Clients und dem (Shipping)-Company–Server.	22.01.2021 – 14:00Uhr
Abnahme Endergebnis • **Live-Funktionstest** mit mehreren Rechnern/Fehlerzuständen in der Schule. • Abgabe **Quellcode** zusammen mit einer **kurzen Beschreibung der einzelnen Klassen** und der dahinterstehenden Überlegungen, • sowie ein ausgefülltes (eigenes) **Testprotokoll** (Testfall, erwartetes Ergebnis, Ergebnis)	17./18. 03.2021

Randbedingungen:

- Bearbeitung erfolgt alleine, d.h. jeder implementiert seine eigene Variante.
- Bearbeitungsfortschritt im Unterricht fließt in die Bewertung mit ein.
- Eigenständige Bearbeitung wird vorausgesetzt. Eine „mittelmäßige" Eigenentwicklung ist besser, als eine kopierte (externe) Lösung! (➔ Abzug!!!)

Bewertungskriterien und Gewichtung:

- Nachvollziehbarer Entwurf (Beziehungen zu den Klassen Thread, Sockets, Streams, sollen daraus ersichtlich sein) 15%
- Konsistenz zwischen Entwurf und Realisierung 5%
- Praxistaugliche Bedienbarkeit (der Test erfolgt an mehreren Rechnern, mit einem SeaTrade-Server und mehreren Schiffen-/Companys) 30%
- strukturierter, verstehbarer Quelltext unter Einhaltung der Java-Konventionen 20%
- Sinnvolle Testfälle-/Testprotokoll 5%
- Integration von Erweiterungen 25%