UML Diagramme

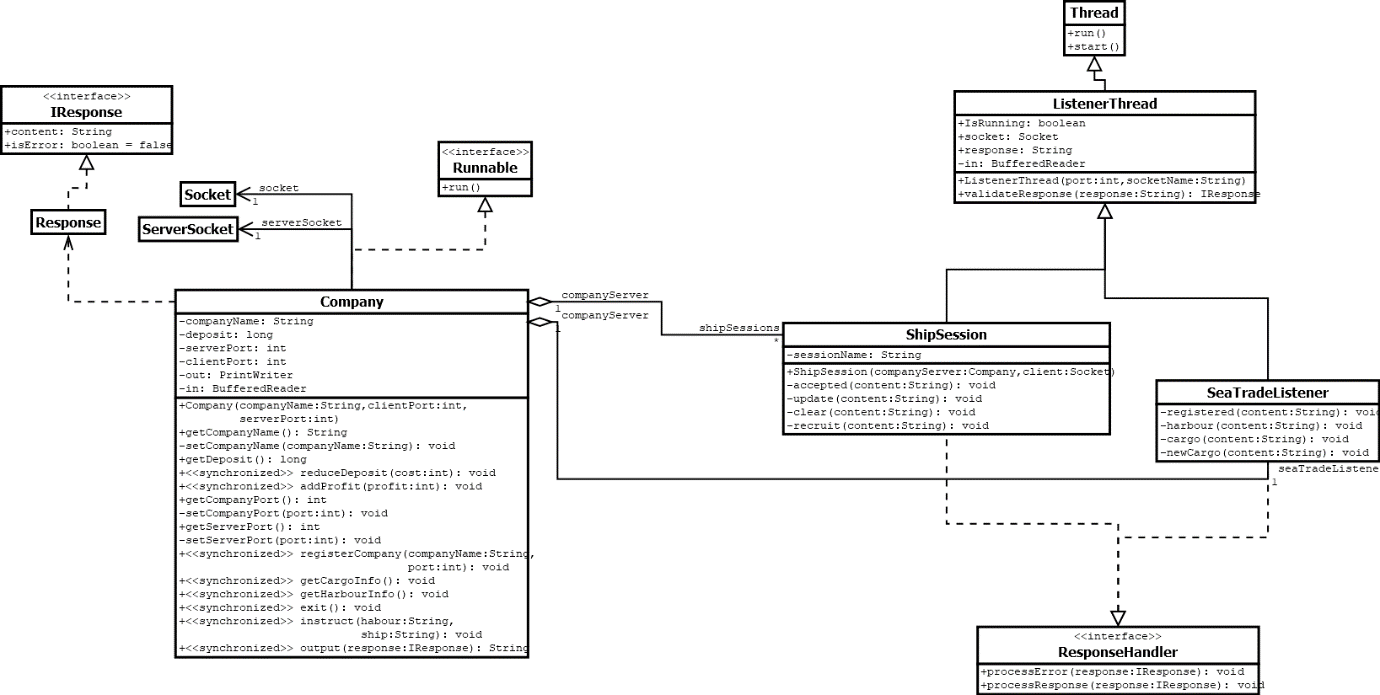
Alle Bilder sind auch im Ordner als PNG zu finden. Sowie die Dia Projektdatei.  


Bild 1: UML Diagramm der Company Klasse

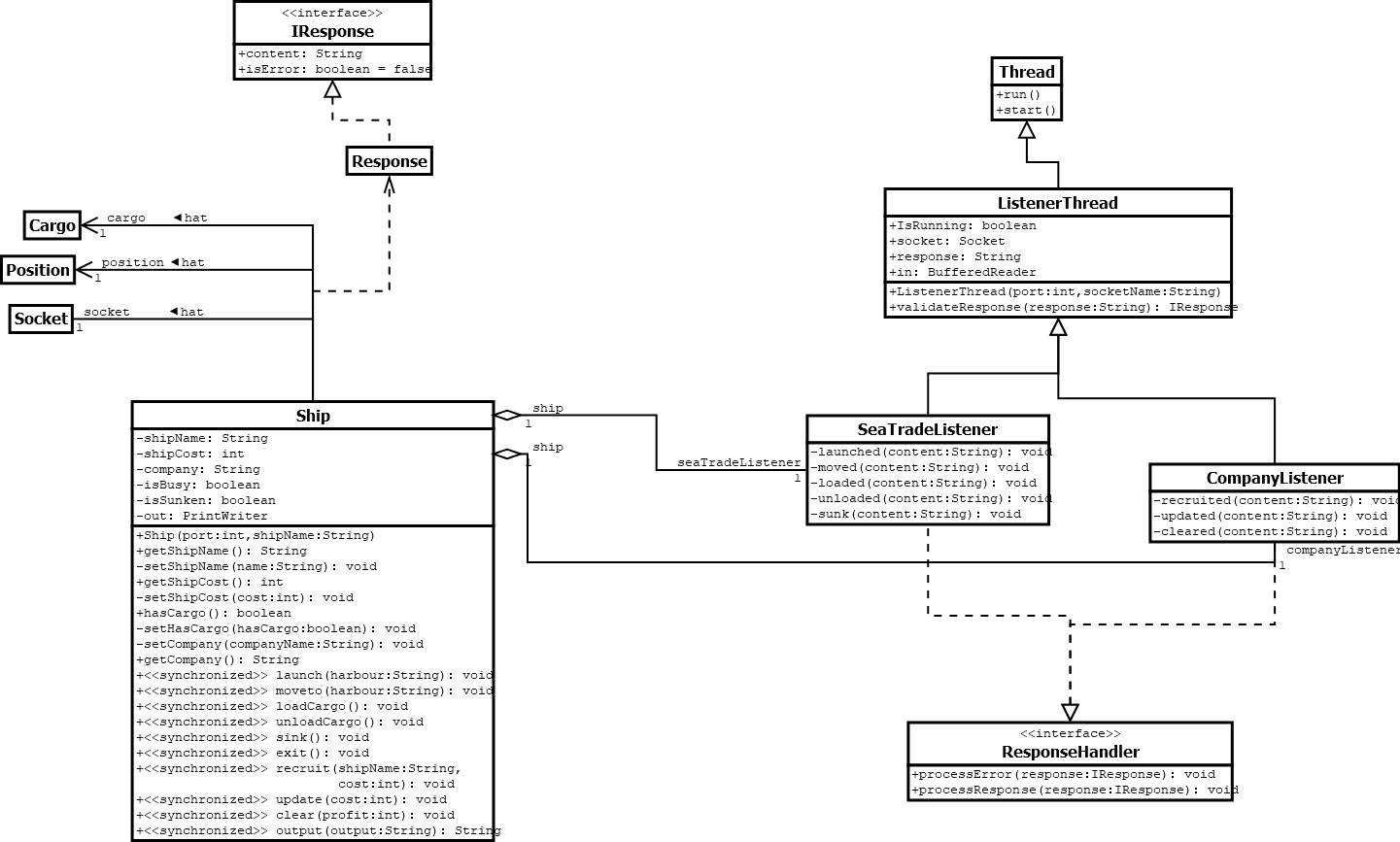


Bild 2: UML Diagramm der Ship Klasse

Kommunikationsprotokoll

Beschreibung des Text-Protokolls zwischen CompanyApp und ShipApp:

S = Ship C= Company

|  |  |  |
| --- | --- | --- |
| Richtung | Funktion | Beschreibung |
| Ship -> Company  S <- C  S <- C | recruit:shipname  recruited:COMPANY|name|deposit:  harbour  error: text | Anfrage, um Schiff der Company hinzuzufügen  Company Infos und Starthafen  Fehler: Nicht genügend Geld |
| C -> S  C <- S  C <- S | instruct:habour:ship  accepted:text  error: text | Schiff beauftragen, Cargo zu holen.  Auftrag angenommen  Fehler: Beschäftigt |
| S -> C  S <- C  S<- C | update:cost  updated: text  error:text | Kosten melden, die durch Bewegung angefallen sind.  Alles ist OK  Fehler: Zu wenig Geld GameOver. |
| S -> C  S <- C  S <- C | clear:profit  cleared:text  error:text | Profit melden.  Alles ist OK  Fehler: - |
| S -> C | exit | Sich von der Company abmelden (freiwillig und gesunken) |

Zusatzinfo/Erklärung eigener Klassen

Response und Response Handler Klasse

Da die Response von jedem Socket ein String mit einem bestimmten Format sein wird, habe ich mir eine Hilfeklasse überlegt. Alle Listener machen aus einem String eine Response diese wird dann validiert. Falls der String ein Error enthält wird der Flag auf True gesetzt. Das Response Handler Interface liefert dann zwei Methoden. Eine, um Errors abzuarbeiten und eine, um gelungene Anfragen abzuarbeiten.

|  |  |  |  |
| --- | --- | --- | --- |
| Funktion | Command | Beispiel Input | Erwartetes Ergebnis |
| Register | register:CompanyName  :SeaTradeServerPort:SeaTradeServerEndpoint  :CompanyServerPort | register:TestCompany  :8150:localhost  :10 |  |
| GetHarbours | harbours: | harbours: |  |

register:TestCompany:8150:localhost:10