UML Diagramme

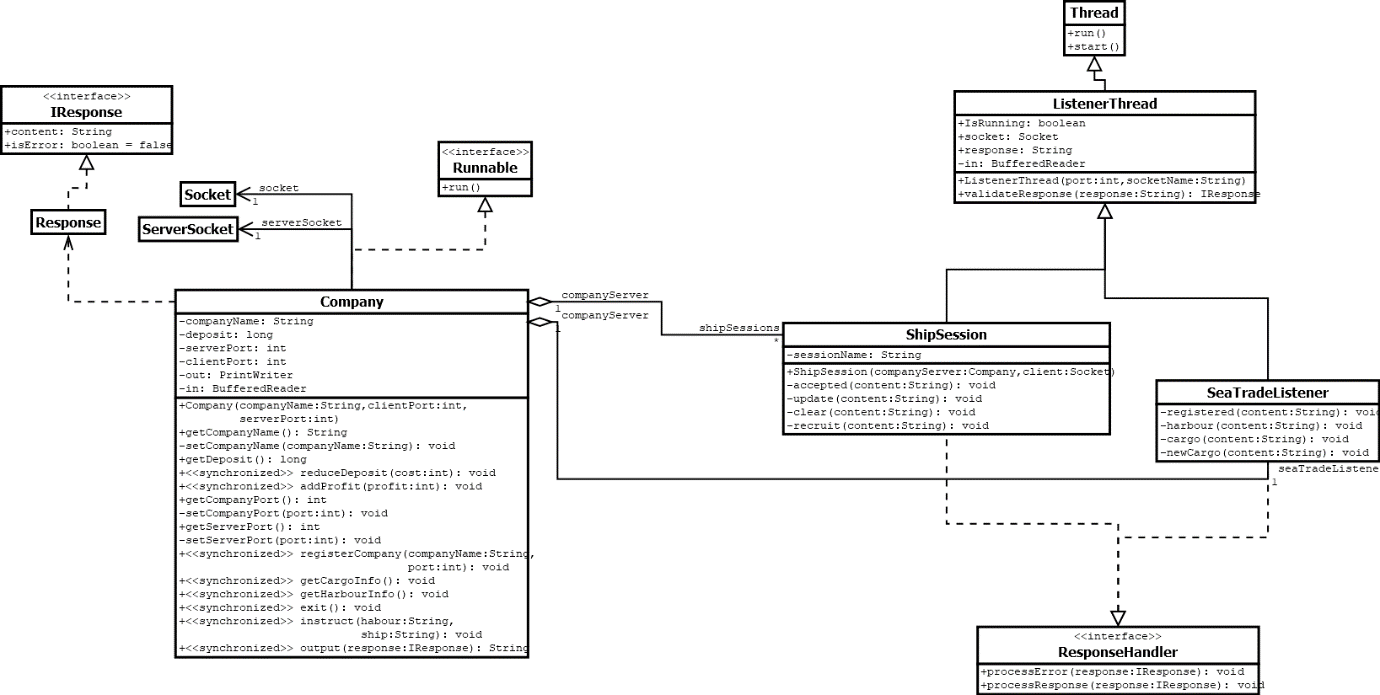
Alle Bilder sind auch im Ordner als PNG zu finden. Sowie die Dia Projektdatei.  


Bild 1: UML Diagramm der Company Klasse

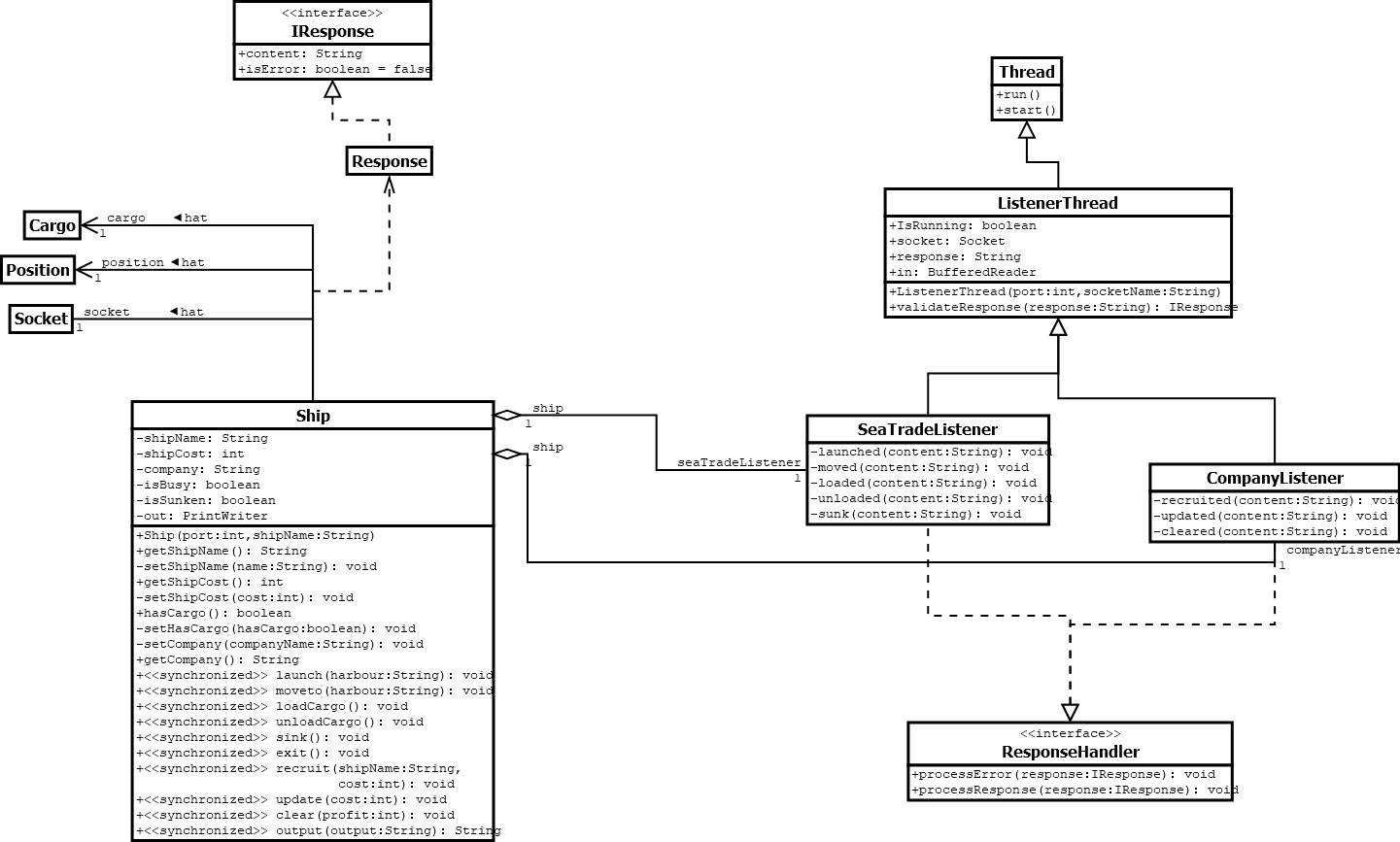


Bild 2: UML Diagramm der Ship Klasse

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  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

Console

Die Console Klassen dienen als Startup für die Applikationen. Jede Konsole besitzt eine StateMachine, einen StateContoller und eine View.

StateMachine

Die StateMachine hilft zusammen mit der BlockingQueue die gleichzeitigen Anfragen zu kontrollieren. Jede StateMachine startet in einem Ready State in dem man Kommandos eingeben kann. Jeder darauffolgende State blockt die Eingaben bis man wieder im Ready State ist.

MessageParser

Die MessageParser Klasse enthält eine BlockingQueue alle Nachrichten die an einem Socket ankommen wird als Message der BlockingQueue hinzugefügt.

MessageListeners

Jeder MessageListener kann sich beim MessageParser auf einen bestimmten MessageType registrieren. Der MessageParser nimmt die oberste Message von der Queue und gibt sie an alle registrierte Listener weiter.

Company

|  |  |  |  |
| --- | --- | --- | --- |
| Funktion | Command | Beispiel Input | Erwartetes Ergebnis |
| Register | register:CompanyName  :SeaTradeServerPort:SeaTradeServerEndpoint  :CompanyServerPort | register:TestCompany  :8150:localhost  :8080 |  |
| GetHarbours | harbours: | harbours: |  |
| GetCargos | cargos: | cargos: |  |
| GetCompany | company: | company: |  |
| instructShip | instruct:harbour:ShipIndex | instruct:halifax:0 |  |
| exit | exit: | exit: |  |

Ship

|  |  |  |  |
| --- | --- | --- | --- |
| Funktion | Command | Beispiel Input | Erwartetes Ergebnis |
| Recruit | recruit:seaTradePort: SeaTradeEndpoint:companyPort: CompanyEndpoint:shipName | recruit:8150:localhost  :8080:localhost:TestShip |  |
| loadCargo | loadcargo: | loadcargo: |  |
| unloadCargo | unloadcargo: | unloadcargo: |  |
| exit | exit: | exit: |  |

Test:

1. Start SeaTrade Server
2. Cargo 10 (SeaTradeServer)
3. Start Company App
4. register:TestCompany:8150:localhost:8080 (Company)
5. harbours: (Company)
6. cargos: (Company)
7. company: (Company)
8. recruit:8151:localhost:8080:localhost:TestShip (Ship)
9. instruct:harbour:0 (company)
10. loadcargo: (ship)
11. instruct:harbour:0 (company)
12. unloadcargo: (ship)
13. exit: (ship)
14. exit: (company)
15. close Server

register:TestCompany:8150:localhost:8080

recruit:8150:localhost:8080:localhost:TestShip