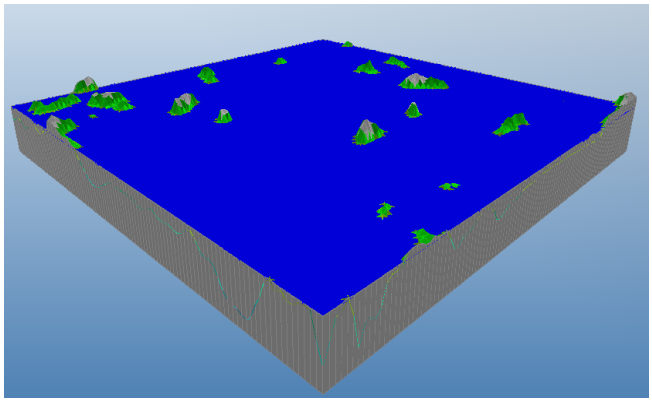
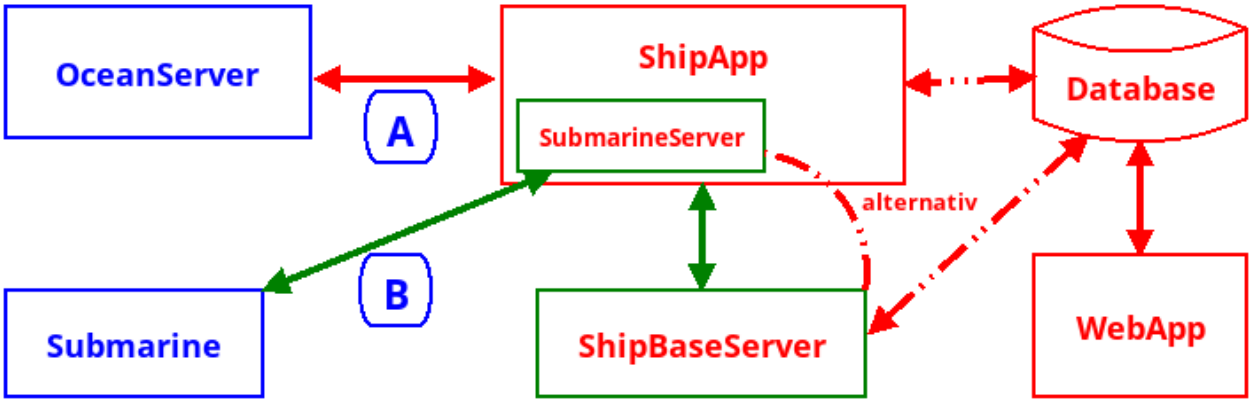


Ein bislang nicht erforschter Meeresbereich soll kartiert und genauer erforscht werden. Dazu werden **Forschungsschiffe** benötigt, die bei Bedarf (bereits vorhandene) **Tauchroboter** aussetzen können, um genauer die Unterwasserwelt zu untersuchen. Gesammelte Daten werden zentral in einer Datenbank gespeichert und der interessierten Öffentlichkeit über Webseiten angeboten.

Der Meeresbereich umfasst ein Gebiet von 10x10km und ist in **Sektoren** (Quadrate) mit einer Kantenlänge von 100m gerastert. Die Koordinaten der Sektoren bewegen sich in X- und Y-Richtung jeweils im Bereich von 0 bis 99. Ein Sektor bildet also eine Fläche von 100x100m ab.



Aufgabe 1 (Basisanforderungen)

- Planen, entwickeln und testen Sie eine Schiffanwendung (**ShipApp**), die sich als Client mit dem vorhandenen Ocean-Server (der den Meeresbereich abbildet) verbindet und Daten sammelt
- **Erweiterung um eine Server-Komponente**:
  - **Variante ‚submarine' :** Erweitern Sie die ShipApp um eine Server-Komponente, die mehrere Tauchroboter-Instanzen (submarines) aussetzen, steuern und die gelieferten Daten sammeln kann.
  - **Variante ‚shipbase':** Erstellen Sie eine eigene ShipBaseServer-Anwendung, die die Daten mehrerer ShipApp Instanzen, die parallel unterwegs sind zusammenführt.
- Alle gesammelten Daten werden in einer zu entwerfenden **Datenbank** zentral gespeichert.
- Eine **Web-Anwendung** bereitet den Datenbestand zur Visualisierung auf.

Aufgabe 2 (Erweiterungen – zur Auswahl, ja nach Neigung, je mehr desto besser)

- Implementierung ShipApp als GUI-Anwendung
- Web-Steuerung der ShipApp
- Fernsteuerung Submarine-Instanzen von der ShipApp aus.
- Algorithmen zur autonomen Schiff-/Submarine-Steuerung zur automatisierten Erforschung
- Speicherung/Visualisierung der gefahrenen Schiffsrouten, gefundener Objekte, ...
- … (andere Features nach Absprache)

Projektablauf:

- Durchführen einer **Anforderungsanalyse** und ableiten von Projektzielen, Anforderungen, gewünschten Ergebnissen.
- Erstellung eines **Pflichtenhefts** (Leistungsbeschreibung des Produkts mit Muss- und Wunsch-Kriterien) inklusive eines **Projektplans**.
- **Entwurf des Grundsystems** unter Anwendung geeigneter OOA/OOD-Hilfsmittel (UML-Diagramme (UseCase-, Activity, State, **Class**- Diagramm), **ERM/RDM** zum Datenbankentwurf).
- OOP-**Implementierung** des Entwurfs inkl. Maßnahmen zur **Qualitätssicherung** (z.B. Tests, Testfälle, …).
- **Präsentation Projektergebnis** und Übergabe einer **Produktdokumentation** (*keine* Projektdokumentation nötig!).

Erwartete Arbeitsergebnisse: E3FI3	**Termin**
Ergebnisse der Anforderungsanalyse: **Pflichtenheft**, **Projektplan**, **OOA/OOD-Entwurf** (mind. UML-Class-Diagramm), **ERM/RDM**	**29.01.2026** 18:00Uhr
Abnahme Endprodukt: Produktpräsentation in einem **Live-Funktionstest.** Abgabe **Quellcode** (mit optionalen Erläuterungen, falls sinnvoll/notwendig) Abgabe ausgefülltes **Testprotokoll** (Testfall, erwartetes Ergebnis, tatsächliches Ergebnis) mit dem das Produkt getestet wurde.	**18.03.2026**

Bewertungskriterien / Gewichtung

	PK-Note	**BFK-Note**
Pflichtenheft, Projektplans	30%	
OOA/OOD Entwurf (UML)	5%	10%
ERM/RDM Entwurf	5%	10%
Projektumsetzung (Kommunikation, Termine, Fortschritt, Arbeitsverteilung)	30%	
Produktabnahme	10%	30%
Integration von Erweiterungen		20%
Strukturiereter, verstehbarer Quelltext (Java-/Web-Komponenten		20%
Testprotokoll mit sinnvollen Testfällen	10%	10%
Produktdokumentation	10%	

Randbedingungen:

- Bearbeitung in 2er/3er Gruppen. Bei 3-er Gruppenwerden entsprechend mehr Erweiterungen gefordert.
- Bearbeitungsfortschritt im Unterricht fließt in die Bewertung ein.
- Eigenständige Bearbeitung wird vorausgesetzt. Eine „mittelmäßige“ Eigenentwicklung ist besser, als eine kopierte (externe) Lösung! (➔ Abzug!!!)
- Es werden Noten für die ganze Gruppe ermittelt. Bei erkennbarer „Ungleichverteilung von Arbeit bzw. Einsatz“ werden individuelle Noten gebildet.

Lieferumfang ocean_explorer_<version>.zip

```
oceanstarter.jar                          # startet oceanserver.jar
oceanserver.jar                           # eigentlicher Server
oceanserver.conf                          # Konfigurationsdatei mit Einstellmöglichkeiten
meer_topo.data, meer_objects.data         # interne Datenmodelle des oceanservers
submarine.jar                             # fertige Tauchroboter-Anwendung
```

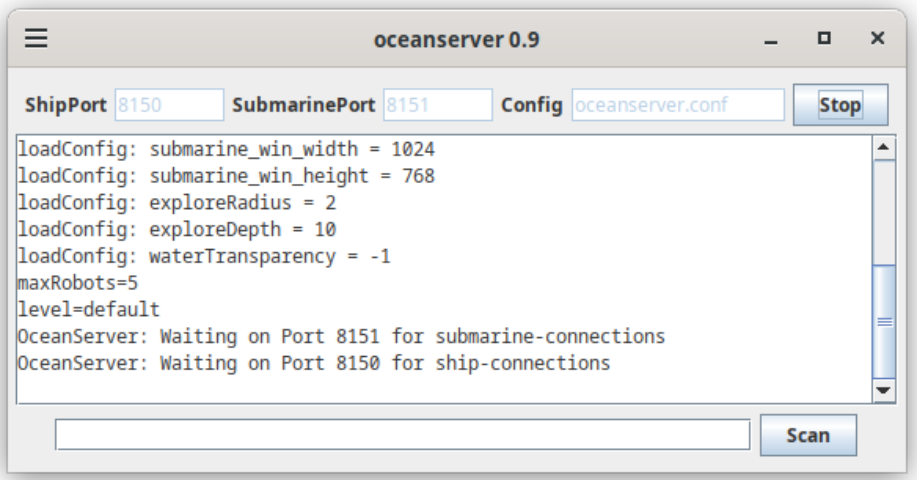
package ocean mit folgenden (Hilfs-)Klassen/Enums:

Allgemein		
`class`	`Vec2D`	Hilfsklasse für 2-dimensionale (Sektor-)Koordinaten [x,y] oder Richtungsvektor eines Schiffs
`class`	`Vec`	Hilfsklasse für 3-dimensionale Koordinaten [x,y,z] in Meter oder Richtunsvektor bei Submarines
`class`	`AppLauncher`	Hilfsklasse mit statischen Methoden zum Start von submarine.jar aus der ShipApp heraus
ShipApp		
`enum`	`Rudder`	Ruderstellung eines Schiffs (Left, Center, Right)
`enum`	`Course`	Kurs (=Schiffsbewegung) (Forward, Backward)
`enum`	`Ground`	Untergrund bei Radarechos (Wasser, Land, ...)
`class`	`RadarEcho`	Bestandteil der Rückmeldung eines Radar(-Requests). Enthält Informationen zu den 8 Sektoren, die sich um die aktuelle Schiffsposition herum befinden.
Submarine		
`enum`	`Route`	Bestandteil von Fernsteuerkommandos (pilot) der ShipApp an submarines. Die Route gibt die Zielrichtung relativ zur aktuellen Submarine-Position/-Ausrichtung an.
`class`	`OceanPicture`	Hilfsklasse mit statischen Methoden zum Umgang mit Bilddaten (Wandlung zur Übertragung in JSON, Laden-/Speichern von PNG-Dateien)

In den Klassen werden teilweise JSON-Klassen aus der org.json-Bibliothek verwendet. Das bedeutet **nicht**, dass dies Bibliothek verwendet werden muss.

Bedienmöglichkeiten Ocean-Server

Nach dem Start öffnet sich ein Admin-Fenster, in dem die Ports für Schiffe und Submarines eingetragen werden können. Diese können auch in der Konfigurationsdatei oceanserver.conf fest hinterlegt werden. In diesem Fall werden die Eingabefelder ignoriert.



Mit dem Start-Button wird der Server gestartet und es öffnet sich eine 3D-Darstellung des zu erforschenden Meeresbereichs. Die verwendeten Konfigurationsparameter werden in der Admin-Console ausgegeben.

Im Eingabefeld können derzeit folgende Befehle eingegeben werden:
verbose : Server gibt mehr Infos im Admin-Fenster aus
refresh oder **r** : 3D-Ansicht „refreshen“
savesunk <dateiname> : Damit können während der Erforschung gesunkene Schiffe/Submarines dauerhaft gespeichert werden. Die geschriebene Datei kann als neues dynamisches Modell (siehe oceanserver.conf) verwendet werden.

Der angezeigte Bereich der 3D-Ansicht kann mit der Bedienleiste oder zugehörigen Tasten verändert werden:

Rotation der Ansicht nach oben (Bild ^) bzw. unten (Bild v)

Rotation der Ansicht nach Links (Pos1) bzw Rechts (Ende)

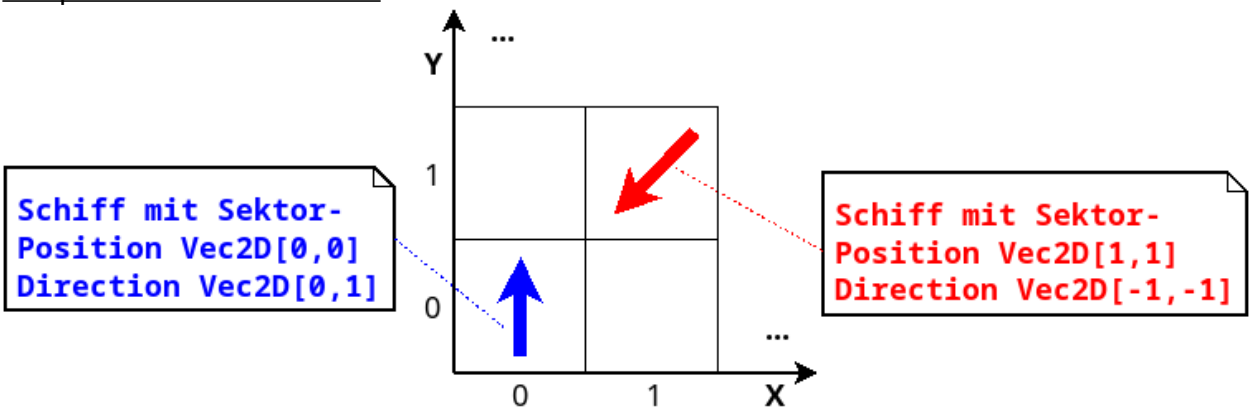
Verschieben des Bildausschnitts mit der Cursortasten

Verkleinern (-) oder Vergrößern (+) des Bildausschnitts

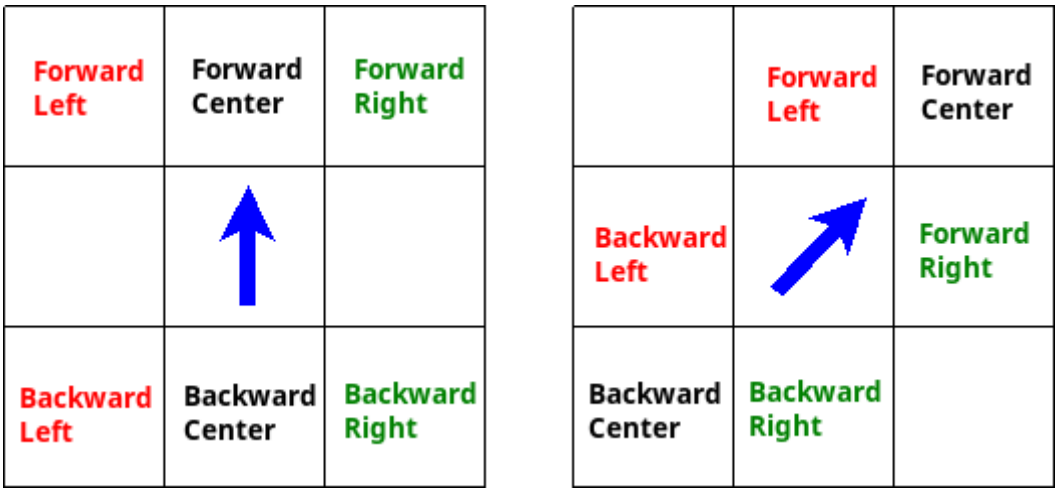
Schiffsnavigation

- Auf einem Sektor darf sich zu einem Zeitpunkt nur ein Schiff befinden.
- Die **Position** wird als 2-dimensionale (Sektor-)Koordinate [x,y] gespeichert (siehe Klasse Vec2D).
- Die **Ausrichtung** (direction) wird durch einen Richtungsvektor angegeben (zulässige Wertebereiche für x und y-Richtung ist jeweils [-1, 0, +1] so dass sich 8 mögliche Richtungen ergeben, die sich um je 45° unterscheiden.

Beispieldaten von 2 Schiffen:



Die **Bewegung** eines Schiffes hängt von der **Ruderstellung** (nach Links, geradeaus, nach Rechts) und dem gewählten **Kurs** (vorwärts oder rückwärts) ab.
Die Abbildung zeigt 2 Beispiele mit den möglichen Zielfeldern je nach Ruderstellung und Kurs:



Jedes Schiff verfügt über ein **Radar** mit dem die Meersoberfläche rund um das Schiff abgefragt werden kann. Das **RadarEcho** enthält dann Infos zu den 8 umliegenden Sektoren um Schiffskollisionen oder das Auflaufen auf Land zu verhindern.

Erforschen eines Sektors

- Von jedem Sektor kann ein **Gesamt-Tiefen-Mittelwert** und die **Varianz** (Standardabweichung) der enthaltenen 10.000 Detail-Tiefenwerte auf Meterbasis abgefragt werden.
- Befinden sich gesunkene Objekte im Sektor kann die Standardabweichung möglichweise nicht genau ermittelt werden. Es kann eine Schwankungsbreite von Messung zu Messung auftreten.

(A) Kommunikationsprotokoll OceanServer (Srv) <=> ShipApp (Client)

Richtung	Cmd	Beschreibung / JSON-Codierung
C => Srv	**Launch**	Einsetzen eines Schiffs mit wählbarem Namen in einen frei wählbaren Sektor und der gewünschten Ausrichtung durch den Richtungsvektor „dir“
	`{ “cmd“:“launch“, “name“:“schiffname“, “typ“:“ship“, “sector“:{ “vec2“:[x,y] }, “dir“:{ “vec2“:[dx,dy] } }`	
Srv => C	**Launched**	Bestätigung eines erfolgreichen Schiff-„launchs“ . Srv sendet eine erzeugte Ship-ID und die genaue Position des Schiffs in absoluten Koordianten [m]
	`{ "cmd":"launched", "id":"ShipID", "abspos":{ "vec2":[xm,ym] } }`	
Srv => C	**Message**	Info- oder Fehlermeldung an den Client
	`{ "cmd":"message", "type“:"info|error", “text“:“Nachrichtentext“ }`	
C => Srv	**Navigate**	Fahren des Schiffs in angrenzenden Sektor, ausgehend von der aktuellen Ausrichtung mit angegebener Ruderstellung (siehe enum Rudder) und Fahrtrichtung (siehe enum Course).
	`{ "cmd":"navigate", "rudder":"Left|Center|Right", "course":"Forward|Backward"}`	
Srv => C	**Move2d**	Neue Schiffsposition nach Navigate-Cmd. Liefert neue Sektor-Koordinaten, neuen Ausrichtungsvektor (dir) und die absolute Schiffsposition in [m]
	`{ "cmd":"move2d","id":"ShipID", "sector":{ "vec2":[x,y] }, "dir":{ "vec2":[dx,dy] }, "abspos":{ "vec2":[xm,ym] } }`	
Srv => C	**Crash**	Schiffsunfall-Benachrichtigung. Liefert Sektor-Koordinaten des gesunkenen Schiffs, und die absolute 3D-Sinkposition in [m]
	`{ "cmd":"crash", "id":"ShipID", "message":"Unfalltext", "sector":{ "vec2":[x,y] }, "sunkPos":{ "vec":[xm,ym,zm] } }`	
C => Srv	**Scan**	Scan des aktuellen Sektors
	`{ "cmd":"scan" }`	
Srv => C	**Scanned**	Rückgabe des Scan-Ergebnisses (mittlere Tiefe des Sektors in ganzen [m] und Standardabweichung (Detail-)Tiefenwerte innerhalb des Sektors als float-Zahl)
	`{ "cmd":"scanned", "id":"ShipID","depth":int,"stddev":float}`	
C => Srv	**Radar**	Radarmessung der umliegenden Sektoren anfordern
	`{ "cmd":"radar" }`	
Srv => C	**Radar-Response**	Ergebnis der Radarmessung. Liefert als Echo eine Liste der 8 umliegenden Sektoren, mit der gemessenen Höhe und der Art des Sektors (Wasser, Land, …, None siehe enum Ground). Sektoren mit Ground.None liegen außerhalb des Meeresbereiches. Bei einer gemessenen Höhe > 0 ist der Sektor nicht befahrbar (z.B. wg. anderem Schiff, Land, …). Es kann **nicht** die Wassertiefe ermittelt werden. (Dazu siehe Scan)
	`{ "cmd":"radarresponse","id":"ShipID", "echos":[ {"sector":{ "vec2":[x,y] }, "height":int, "ground":"GROUND" }, … ] }`	
C => Srv	**Exit**	Schiff aus dem Spiel nehmen, inklusive eventuell ausgesetzer Tauchroboter
	`{ "cmd":"exit" }`	

Nutzung von Tauchrobotern (nur bei Server-Variante ‚submarine')

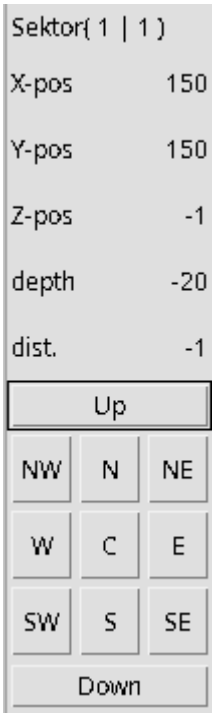
Voraussetzung in der ShipApp

- Die ShipApp muss einen **ServerSocket bereitstellen**, mit dem sich der Tauchroboter (=submarine) verbindet, um Daten auszutauschen. Idealerweise sollte die ShipApp **mehrere Submarine-Sessions verwalten** können.
- Bitte Beachten: Gibt es mehrere Schiffe auf dem Meer, muss jede Instanz einen eigenen Server-Port verwenden.

Aussetzen Submarine

- Ein Schiff hat 4 Tauchroboter an Bord, die innerhalb eines Sektors ausgesetzt werden können, um Detailinformationen dieses Sektor zu erforschen.
- Das **Aussetzen** erfolgt durch den **Start des fertigen submarine.jar** -Archives mittels der Klasse **AppLauncher**.
- Die **Kommunikation** zwischen Schiff und Submarine erfolgt über das **Protokoll (B)** auf der nächsten Seite.

Tauchvorgang



- Im Bedienpanel wird der **aktuelle Sektor**, in dem getaucht wird angezeigt.
- Außerdem ist die **absolute 3D-Position** (Vec) des Tauchroboters im Meer abzulesen.
- Es wird auch die **absolute Meerstiefe (depth)** unterhalb des Tauchroboters gemessen und angezeigt
- Als Navigationshilfe wird auch der **Abstand in Tauchrichtung angezeigt**. Der Wert -1 bedeutet, dass kein absehbares Hindernis vorhanden ist. Bei Werten >0 wird angezeigt, wie weit [m] der Roboter vom Meeresboden noch entfernt ist.
- Über das Bedienpanel kann sich der **Tauchroboter** innerhalb des Sektors unter Wasser (im 3D-Raum **bewegen**).
  - Up/Down: senkrechtes auf-/absteigen um einen Meter im Wasser
  - NW,N,NE, … : Bewegt den Roboter um eine Einheit in Tauchrichtung und ändert diese relativ zur gewünschten „Himmelsrichtung".
  - Der Rauchroboter kann nur zwischen +/- 45° geneigt sein, also nicht senkrecht nach unten/oben oder Überkopf ausgerichtet werden.
- Über das „**Pilot**" Kommando kann der Roboter vom Schiff aus **ferngesteuert** werden.
- Der Tauchroboter liefert bei Änderungen seinen **aktuellen Status** mit der „**Ready**" Nachricht.
- Taucht der Roboter nah genug über dem Meeresboden, werden absolute Meereskoordinaten in Meterauflösung erfasst und als „**Measure**"- Liste an die ShipApp gesendet. Dabei wird ein Punkt nur einmal gesendet, auch wenn er mehrfach „übertaucht" wird. Übertragene Punkte werden in der Roboteransicht rot gekennzeichnet.
- Zu jedem Zeitpunkt kann von der aktuellen Ansicht des Roboters ein **Foto** gemacht werden, das als „**Picture**" an die ShipApp weitergeleitet wird.

Aufnehmen Submarine

Taucht der Roboter wieder auf (Z-Pos >0) und befindet sich nah genug am Schiff, so wird er automatisch vom Schiff wieder aufgenommen (=> Nachricht „**Arise**" an ShipApp).

Während des Tauchvorgangs darf das Schiff nicht den Sektor wechseln, da sonst die Verbindung zu den Tauchrobotern abbricht und diese verloren gehen (sinken).

(B) Kommunikationsprotokoll ShipApp (Srv) <=> Submarine (Client)

ShipApp kann mittels der mitgelieferten Klasse **`AppLauncher`** eine Tauchroboter-Instanz (= eigene JavaVM, die **`submarine.jar`** ausführt) starten. Die `startSubmarine`-Methoden benötigen zwingend folgende Parameter:

`String shipID`	Schiffsidentifikation, die der OceanServer dem Schiff vergeben hat (siehe CmdLaunched).
`String shipHost`	Hostname auf dem die ShipApp läuft
`int shipPort`	Portnummer des ServerSockets, auf dem die ShipApp auf Submarine-Verbindungen wartet
`String oceanSrvHost`	Hostname auf dem der Ocean-Server läuft
`int oceanSrvPort`	Submarine-Port des Ocean-Servers

Nach Verbindung von Submarine und ShipApp können folgende Kommandos genutzt werden:

Richtung	**Cmd**	**Beschreibung / JSON-Codierung**
C => Srv	**Ready**	Statusmeldung des Tauchroboters. Geliefert wird die aktuelle 3D-Positiion im Ozean, der aktuelle Ausrichtungsvektor (dir), die absolute Wassertiefe an der aktuellen x/y-Position und die Distanz [m] in Fahrtrichtung zum nächsten Hindernis (distance -1 bedeutet kein Hindernis in Sicht).
	`{ "cmd":"ready","id":"submarineID", "pos":{ "vec":[x,y,z] }, "dir":{ "vec":[dx,dy,dz] }, "depth": int, "distance": int }`	
Srv => C	**Pilot**	Fernsteuerung des Tauchroboters. Zur Bewegungssteuerung dienen die Werte des `enum Route { C, N, NE, …, UP, DOWN }`. Der Tauchroboter bewegt sich von aktueller Position und Ausrichtung anhand der gesendeten Route-Werte. Bei `Route.None` wird der action-String ausgewertet: `"take_photo":` löst eine Foto-Aufnahme des Tauchroboter-Sichtfeldes aus. `"locate":` versucht Objekte in der Nähe zu orten
	`{ "cmd":"pilot", "route":"ROUTE", "action":"..." }`	
C => Srv	**Message**	Nachricht an ShipApp mit folgenden Typen: „**error**"(=Fehlernachricht), „**info**" (= allgemeine Information), „**located**" (=Ortung (locate) erfolgreich)
	`{ "cmd":"message", "type":"...", "text":"Nachrichtentext", "pos":{ "vec":[x,y,z] } }` *`# "pos" nur bei type == located`*	
C => Srv	**Measure**	Während der Tauchfahrt erfasst der Tauchroboter detaillierte Tiefenwerte, sofern sie im Bereich seiner Sensorik liegen. Gibt es nach einer Bewegung neue Werte (die noch nicht gesendet wurden) überträgt er die Messwerte als „vecs"-Liste von absoluten 3D-Koordinaten.
	`{"cmd":"measure", "vecs":[ [x1,y1,z1], … ,[xn,yn,zn] ] }`	
C => Srv	**Picture**	Bildübertragung (PNG). Siehe Hilfsklasse `OceanPicture`
	`{ "cmd":"picture", "id":"submarineID", "pos":{ "vec":[x,y,z] },"dir":{ "vec":[dx,dy,dz ] }, "picture":"PNG in Hex-String codiert" }`	
C => Srv	**Crash**	Submarine-Tauchunfall-Benachrichtigung. Liefert Sektor-Koordinaten des gesunkenen Tauchroboters, und die absolute 3D-Sinkposition in [m]
	`{ "cmd":"crash", "id":"ShipID", "message":"Unfalltext", "sector":{ "vec2":[x,y] }, "sunkPos":{ "vec":[xm,ym,zm] } }`	
C => Srv	**Arise**	Submarine ist wieder aufgetaucht und wird vom Schiff aufgenommen. Submarine-Anwendung beendet sich automatisch.
	`{ "cmd":"arise","id":"submarineID", "arisePos":{ "vec":[x,y,z] } }`	