LogWriter Java Documentation

Sascha Just

1. September 2014

Inhaltsverzeichnis

Kapitel 1

Benutzung des Sopra-Loggers

Der Sopra-Logger stellt eine Log-Implementierung zur Verfügung welche von Ihrem Simulator genutzt werden muss. Das produzierte Log aus ihrem Simulator wird mit dem Log der Referenzimplemntierung verglichen, um ihre Implementierung auf Korrektheit zu überprüfen. Die Interna des von uns bereitgestellten Loggers sind dabei nicht von Belang.

Wir raten Ihnen dazu, eine eigene Implementierung der `LogWriter` Schnittstelle anzufertigen. Dies ists sowohl zu Debugzwecken hilfreich, als auch bei der Implementierung Ihrer GUI. Optionale (nicht bestehensrelevante) wie etwa zusätzliche Software (wie z.B. Debugger, Hochsprachencompiler, Karteneditoren, etc...) können in der Programmiersprache ihrer Wahl erstellt werden. Hier bietet es sich evtl. an ein Logformat zu wählen welches leicht mit anderen Infrastrukturen geteilt werden kann, wie etwa JSON[1].

1.1 Benutzung des Loggers

Vor der Nutzung des Loggers muss dieser zunächst initialisiert werden. Dies ist notwendig, um dem Logger mitzuteilen, wohin die Daten geschrieben werden und wie der initiale Zustand der Welt aussieht.

1.1.1 Initialisierung

1. Nach dem Erstellen der Loggerinstanz muss zuerst die `init(...)` Methode aufgerufen werden.
2. Nun müssen dem Logger alle Kacheln der Weltkarte mitgeteilt werden. Dies geschieht mittels der Methode `addCell(...)`.
3. Im nächsten Schritt werden alle Schiffe und Gegenstände geloggt, welche sich zu Beginn des Spiels auf der Karte befinden. Hierzu dient die `create(...)` Methode.
4. Zusätzlich müssen die Punktzahlen der Flotten initialisiert werden. Ausgehend davon, dass kein Handycap vorhanden ist, werden alle Flottenpunkte mit 0 initialisiert. Dies geschieht mittels der `fleetScore(...)` Methode.
5. Da das Log nun über alle relevanten Daten verfügt, muss der initiale Zustand geloggt werden. Hierzu ziehen Sie die Methode `logStep()` hinzu.

[1] http://www.json.org

Die Initialisierung ist damit abgeschlossen.

1.1.2 Spielverlauf

Im Laufe der Simulation müssen dem Logger alle Änderungen im Zustand des Spiels mitgeteilt werden. Dies geschieht mit den Methoden `create(...)`, `destroy(...)` und `notify(...)`. Zudem bietet sich die Möglichkeit mehrere Änderungen am Spielstatus mittels Transaktionen gruppiert an den Logger zu senden. Dies ist mit den Methoden `beginTransaction(...)` und `commitTransaction(...)` realisiert.

Am Ende jedes Zugs muss dem Logger mittels `logStep()` signalisiert werden, dass die Runde vorbei ist.

1.1.3 Abschluss der Simulation

Ist die Simulation beendet (und somit das letzte Mal `logStep()` aufgerufen worden) muss der Logger geschlossen werden. Dies ist unbedingt notwendig, da die `logStep()` Methode nicht garantiert, dass alle Daten herausgeschrieben werden. Erst bei Aufruf der Methode `close()` werden die internen Puffer geleert und alle Resourcen freigegeben. Resourcen welche von außen an den Logger weitergegeben wurden, werden dabei nicht geschlossen/freigegeben.

Klassenhierarchie

Klassen

- java.lang.Object
  - java.lang.Enum
    - de.unisaarland.cs.st.pirates.logger.LogWriter.Cell `Cell` (in 2.3, Seite 15)
    - de.unisaarland.cs.st.pirates.logger.LogWriter.Entity `Entity` (in 2.4, Seite 16)
    - de.unisaarland.cs.st.pirates.logger.LogWriter.Key `Key` (in 2.5, Seite 17)

Schnittstellen

- de.unisaarland.cs.st.pirates.logger.LogWriter `LogWriter` (in 2.1, Seite 6)
- de.unisaarland.cs.st.pirates.logger.LogWriter.Transaction `Transaction` (in 2.2, Seite 14)

Kapitel 2

Paket de.unisaarland.cs.st.pirates.logger

Paket Inhalte *Seite*

Schnittstellen

Klassen

2.1 Schnittstelle LogWriter

Die Schnittstelle des LogWriters im Software Praktikum 2014.

Diese Schnittstelle spezifiziert die Interaktion mit dem Logger für die Aufgabenstellung Pirates of the Saaribbean. Eine Erweiterung dieser Schnittstelle in Ihrer Implementierung ist durchaus möglich. Erstellen Sie dazu eine neue Schnittstelle, welches diese erweitert. Bedenken Sie, dass Sie im Standardfall mit der von uns bereitgestellten Loggerimplementierung loggen müssen. Sollten Sie die LogWriter-Schnittstelle erweitern ist es also notwendig die Implementierung in einer neuen Klasse zu kapseln.

2.1.1 Deklaration

public interface LogWriter

2.1.2 Methoden Überblick

addCell(LogWriter.Cell, Integer, int, int) Fügt dem Log eine Kachel hinzu.
addCustomHeaderData(String) Erlaubt das Hinzufügen eines beliebigen Textfeldes zum Header des Logs.
beginTransaction(LogWriter.Entity, int) Beginnt eine neue Transaktion.
close() Schließt den LogWriter.
commitTransaction(LogWriter.Transaction) Übergibt die Transaktion an den Logger.
create(LogWriter.Entity, int, LogWriter.Key[], int[]) Erstellt ein neues Objekt.
destroy(LogWriter.Entity, int) Zerstört ein Objekt im Spiel.
fleetScore(int, int) Signalisiert die Änderung des Punktestands einer Flotte (z.B. beim Abliefern von Schätzen in einer Piratenbasis).
init(OutputStream, String, String[]) Initialisiert den LogWriter.
logStep() Signalisiert dem Logger das Ende einer Runde.
notify(LogWriter.Entity, int, LogWriter.Key, int) Benachrichtigt den Logger über eine Änderung im Spielgeschehen.

2.1.3 Methoden

- **addCell**
  `LogWriter` **addCell**`(LogWriter.Cell` **type**`, java.lang.Integer` **affiliation**`, int` **x**`, int` **y**`) throws java.lang.NullPointerException, java.lang.ArrayIndexOutOfBoundsException, java.lang.IllegalArgumentException, java.lang.IllegalStateException`

  – **Beschreibung**
  Fügt dem Log eine Kachel hinzu.

  Diese Methode dokumentiert eine Kachel in der Logger Implementierung. Nachdem der Logger initialisiert wurde (siehe `init` (in 2.1.3, Seite 12)) müssen alle Kacheln der Karte dem Log hinzugefügt werden. Der Aufruf dieser Methode ist nur gültig *nach* dem Aufruf der `init(...)` Methode und *vor* dem ersten Aufruf von `logStep` (in 2.1.3, Seite 13).

  Es stehen 3 Arten von Kacheln zur Verfügung:

  * Wasser (`WATER` (in 2.3.4, Seite 15)),
  * Inseln (`ISLAND` (in 2.3.4, Seite 15)) und
  * Versorgungsinsel (`SUPPLY` (in 2.3.4, Seite 15)).

  Schätze, welche auf der Karte liegen werden im nächsten Schritt mit der `create` (in 2.1.3, Seite 10) Methode hinzugefügt.
  Beispiel einer 2x2 Karte:

```
# 9
 a .
```

Das Hinzufügen dieser Karte zum Log ist folgendermaßen umzusetzen:

```
logger.addCell(Cell.ISLAND, null, 0, 0)
logger.addCell(Cell.WATER, null, 0, 1)
logger.create(Entity.Treasure, 0, new Key[] { X_COORD, Y_COORD, VALUE }, new int[]
{ 0, 1, 9 });
logger.addCell(Cell.WATER, 0, 1, 0)
logger.addCell(Cell.WATER, null, 1, 1)
```

- **Parameter**
  - `type` – Der Typ der Kachel (siehe `Cell` (in 2.3, Seite 15)).
  - `affiliation` – Im Falle einer Piratenbasis gibt dieser Wert die Flotte an. Flotten beginnen bei 0 (entspricht 'a'). Ist die Kachel keine Piratenbasis wird dieser Wert auf `null` gesetzt.
  - `x` – Die x-Koordinate der Kachel wie in der Aufgabenstellung beschrieben.
  - `y` – Die y-Koordinate der Kachel wie in der Aufgabenstellung beschrieben.
- **Gibt zurück** – den Logger
- **Wirft**
  - `java.lang.NullPointerException` – wenn der Typ der Kachel nicht gesetzt wurde.
  - `java.lang.ArrayIndexOutOfBoundsException` – wenn die Identität der Flotte (`affiliation`) die Anzahl der Angegebenen Programme in `init` (in 2.1.3, Seite 12) übersteigt.
  - `java.lang.IllegalArgumentException` – sofern entweder `x` oder `y` negativ sind.
  - `java.lang.IllegalStateException` – wenn die Methode nicht zwischen `init` (in 2.1.3, Seite 12) und dem ersten `logStep` (in 2.1.3, Seite 13) aufgerufen wurde.

- **addCustomHeaderData**
  `LogWriter` **addCustomHeaderData**`(java.lang.String data) throws java.lang.NullPointerException, java.lang.ArrayIndexOutOfBoundsException`

  - **Beschreibung**
    Erlaubt das Hinzufügen eines beliebigen Textfeldes zum Header des Logs.

    Beispiel:

    ```
    logger.addCustomHeader(meineHochsprachenprogramme);
    ```

  - **Parameter**
    - `data` – Gibt den String an, der dem Header hinzugefügt werden soll.
  - **Gibt zurück** – den Logger.
  - **Wirft**
    - `java.lang.NullPointerException` – sofern `data` nicht gesetzt ist.
    - `java.lang.ArrayIndexOutOfBoundsException` – sofern `data` eine Länge von 1.000.000 überschreitet.

    * `java.lang.IllegalStateException` – wenn die Methode nicht zwischen `init` (in 2.1.3, Seite 12) und dem ersten `logStep` (in 2.1.3, Seite 13) aufgerufen wurde.

- **beginTransaction**
  `LogWriter.Transaction` **beginTransaction**`(LogWriter.Entity` **entity**`, int` **id**`) throws java.lang.NullPointerException, java.lang.IllegalArgumentException, java.lang.IllegalStateException`

  – **Beschreibung**
  Beginnt eine neue Transaktion.

  Transaktionen gruppieren Änderungsbenachrichtigungen (Deltas), um das Kommunizieren mit dem Logger etwas handlicher zu gestalten. Der Effekt auf den Logger ist jedoch kein anderer als das sequentielle Aufrufen von `notify` (in 2.1.3, Seite 13).

  Beachten Sie hierbei, dass die Reihenfolge in der sie unterschiedliche Tags setzen keine Rolle spielt. Das Log wird *rundenbasiert* geschrieben. Der letzte gesetzte Wert ist also ausschlaggebend.

  Folgende Programmausschnitte erzeugen ein identisches Log:

```
logger.notify(Entity.SHIP, 12, Key.MORAL, 1);
logger.notify(Entity.SHIP, 12, Key.DIRECTION, 2);
logger.notify(Entity.SHIP, 12, Key.PC, 17);

Transaction t = logger.beginTransaction(Entity.SHIP, 12);
t.set(Key.MORAL, 0);
t.set(Key.DIRECTION, 3);
t.set(Key.DIRECTION, 2);
t.set(Key.MORAL, 2);
t.set(Key.PC, 17);
logger.commitTransaction(t);
```

  – **Parameter**
    * `entity` – der Objekttyp (siehe `Entity` (in 2.4, Seite 16))
    * `id` – die Identität des Objekts. Dies stellt zusammen mit dem Objekttyp den Schlüssel des Objekts im Log da.
  – **Gibt zurück** – eine neue Transaktion mit genanntem Schlüssel (garantiert nicht `null`).
  – **Wirft**
    * `java.lang.NullPointerException` – sofern `entity` nicht gesetzt ist.
    * `java.lang.IllegalArgumentException` – falls `id` negativ ist.
    * `java.lang.IllegalStateException` – sollte die Methode vor `init` (in 2.1.3, Seite 12) aufgerufen worden sein.
  – **Siehe auch**
    * `LogWriter.Transaction` `Transaction` (in 2.2, Seite 14)

- **close**
  `void` **close**`() throws java.lang.IllegalStateException, java.io.IOException`
  - **Beschreibung**
    Schließt den LogWriter.

    Das Schließen des LogWriters schreibt die möglicherweise verbleibenden Daten aus dem Puffer in den , welcher in der `init` (in 2.1.3, Seite 12) Methode angegeben wurde und gibt alle internen Resourcen frei. **Die Methode schließt *nicht* den .**
  - **Wirft**
    * `java.lang.IllegalStateException` – sollte die Methode vor dem ersten `logStep` (in 2.1.3, Seite 13) aufgerufen werden.
    * `java.io.IOException` – sollte das Schreiben auf den fehlschlagen.

- **commitTransaction**
  `LogWriter` **commitTransaction**`(LogWriter.Transaction` **transaction**`) throws java.lang.NullPointerException, java.lang.IllegalArgumentException, java.lang.IllegalStateException`
  - **Beschreibung**
    Übergibt die Transaktion an den Logger.
  - **Parameter**
    * `transaction` – die Transaktion, welche die Änderungen (Deltas) enthält, welche geloggt werden sollen.
  - **Gibt zurück** – den Logger
  - **Wirft**
    * `java.lang.NullPointerException` – falls `transaction` nicht gesetzt ist.
    * `java.lang.IllegalArgumentException` – sofern die Transaktion keine Daten enthält.
    * `java.lang.IllegalStateException` – sofern die Methode vor dem ersten `logStep` (in 2.1.3, Seite 13) aufgerufen wurde.
  - **Siehe auch**
    * `LogWriter.Transaction` `Transaction` (in 2.2, Seite 14)
    * `LogWriter.beginTransaction(LogWriter.Entity,int)` `beginTransaction` (in 2.1.3, Seite 9)

- **create**
  `LogWriter` **create**`(LogWriter.Entity` **entity**`, int` **id**`, LogWriter.Key[]` **keys**`, int[]` **values**`) throws java.lang.NullPointerException, java.lang.IllegalArgumentException, java.lang.ArrayIndexOutOfBoundsException, java.lang.IllegalStateException`
  - **Beschreibung**
    Erstellt ein neues Objekt.

    Dies ist sowohl erforderlich für Schiffe, als auch für alle Gegenstände im Spiel.

Beispiel: Es wird eine Boje der Flotte b mit Kodierung 4 auf der Kachel 2/7 abgesetzt. Die höchste Gegenstandsidentität zu diesem Zeitpunkt ist 112.
`logger.create(Entity.BUOY, 113, new Key[] { FLEET, VALUE, X_COORD, Y_COORD }, new int[] { 1, 4, 2, 7 });`

- **Parameter**
  - `entity` – Der Objekttyp des zu erstellenden Objekt. Siehe: `Entity` (in 2.4, Seite 16)
  - `id` – Die Identität des Objekts, welche zusammen mit dem Typ ein Objekt eindeutig identifiziert.
  - `keys` – Die Schlüssel, welche Angeben welche Werte gesetzt werden. Siehe: `Key` (in 2.5, Seite 17)
  - `values` – Die Werte, welche gesetzt werden in entsprechender Reihenfolge zu den in `keys` spezifizierten Schlüsseln.
- **Gibt zurück** – Den Logger.
- **Wirft**
  - `java.lang.NullPointerException` – sofern `entity`, `keys` oder `values` nicht gesetzt sind.
  - `java.lang.IllegalArgumentException` – falls `id` negativ ist oder ein Schlüssel angegeben wurde, welcher für den durch `entity` spezifizierten Objekttyp nicht zur Verfügung steht.
  - `java.lang.ArrayIndexOutOfBoundsException` – im Falle, dass die Länge des `keys` Feldes nicht mit der Länge des `values` übereinstimmt.
  - `java.lang.IllegalStateException` – falls die Method vor der `init` (in 2.1.3, Seite 12) Methode aufgerufen wird.

- **destroy**
  `LogWriter` **`destroy`**`(LogWriter.Entity` **`entity`**`, int` **`id`**`) throws java.lang.NullPointerException, java.lang.IllegalArgumentException, java.lang.IllegalStateException`

  - **Beschreibung**
    Zerstört ein Objekt im Spiel.

    Dies ist unter anderem erforderlich beim Sinken von Schiffen, aufnehmen von Schätzen, welche von den Kacheln entfernt werden, oder aber auch beim aufnehmen von Bojen.

    Beispiel: Das Schiff mit der Identität 13 sinkt.
    `logger.destroy(Entity.SHIP, 13);`

  - **Parameter**
    - `entity` – Der Objekttyp des zu zerstörenden Objekts. Siehe: `Entity` (in 2.4, Seite 16).

    * `id` – Die Identität des zu zerstörenden Objekts.
  - **Gibt zurück** – Den Logger.
  - **Wirft**
    * `java.lang.NullPointerException` – sofern `entity` nicht gesetzt ist.
    * `java.lang.IllegalArgumentException` – falls `id` negativ ist.
    * `java.lang.IllegalStateException` – falls die Methode vor dem ersten `logStep` (in 2.1.3, Seite 13) aufgerufen wird.

- **fleetScore**
  `LogWriter` **fleetScore**`(int` **id**`, int` **value**`) throws java.lang.IllegalArgumentException, java.lang.IllegalStateException`

  - **Beschreibung**
    Signalisiert die Änderung des Punktestands einer Flotte (z.B. beim Abliefern von Schätzen in einer Piratenbasis).
  - **Parameter**
    * `id` – Die Identität der Flotte, deren Punktestand geändert werden soll. Flottenidentitäten beginnen bei 0, welches dem Alias *a* auf einer Karte entspricht.
    * `value` – Die tatsächliche Punktzahl der Flotte. Dies ist der *absolute* Punktestand und stellt nicht die Änderung dar.
  - **Gibt zurück** – Den Logger.
  - **Wirft**
    * `java.lang.IllegalArgumentException` – Falls `id` oder `value` negativ sind.
    * `java.lang.ArrayIndexOutOfBoundsException` – Falls `id` größer als die Anzahl der spezifizierten Taktikprogramme ist.
    * `java.lang.IllegalStateException` – Falls die Methode vor `init` (in 2.1.3, Seite 12) aufgerufen wird.

- **init**
  `void` **init**`(java.io.OutputStream` **logStream**`, java.lang.String` **map**`, java.lang.String[]` **programs**`) throws java.lang.NullPointerException, java.io.IOException, java.lang.ArrayIndexOutOfBoundsException`

  - **Beschreibung**
    Initialisiert den LogWriter.

    Dies ist die erste Methode, die nach dem Erstellen des Loggers aufgerufen werden muss. Hier wird dem Logger mitgeteilt wohin er loggen soll und es wird eine Textrepräsentation von Karte und Taktikprogrammen übergeben, welche im Header des Logs verwendet werden.

    Beispiel:
    ```
    String map = readFile(...);
    String programs = readPrograms(...);
    logger.init(System.err, map, programs);
    ```

  - **Parameter**
    * `logStream` – Der auf welchen der Logger schreibt. Im Standardfall ist dies ein OutputStream auf die angegebene Logdatei.
    * `map` – Eine String-Repräsentation der Weltkarte.
    * `programs` – Die Taktikprogramme in String-Repräsentation.
  - **Wirft**
    * `java.lang.NullPointerException` – Falls eines der Argument nicht gesetzt ist;
    * `java.io.IOException` – Falls der `logStream` nicht beschreibbar ist.
    * `java.lang.ArrayIndexOutOfBoundsException` – Falls das `programs` Feld leer ist.

- **logStep**
  `void` **logStep**`() throws java.lang.IllegalStateException, java.io.IOException`

  - **Beschreibung**
    Signalisiert dem Logger das Ende einer Runde.

    Dies führt nicht notwendigerweise dazu, dass umgehend Daten in den geschrieben werden. Erst der Aufruf von `close` (in 2.1.3, Seite 10) am Ende der Simulation garantiert das Rausschreiben aller Daten im internen Puffer.
  - **Wirft**
    * `java.lang.IllegalStateException` – Falls zuvor nicht `init` (in 2.1.3, Seite 12) aufgerufen wurde.
    * `java.io.IOException` – Falls das Schreiben auf den fehlschlägt.

- **notify**
  `LogWriter` **notify**`(LogWriter.Entity` **entity**`, int` **id**`, LogWriter.Key` **key**`, int` **value**`) throws java.lang.NullPointerException, java.lang.IllegalArgumentException, java.lang.IllegalStateException`

  - **Beschreibung**
    Benachrichtigt den Logger über eine Änderung im Spielgeschehen. Hiermit lässt sich pro Aufruf exakt eine Änderung eines Wertes eines Spielobjekts kodieren.
  - **Parameter**
    * `entity` – Der Typ des Objekts welches die Änderung erfährt. Siehe: `Entity` (in 2.4, Seite 16).
    * `id` – Die Identität des Objekts.
    * `key` – Der Schlüssel des Wertes, welcher von der Änderung betroffen ist. Siehe `Key` (in 2.5, Seite 17).
    * `value` – Der zu dem Schlüssel gehörige Wert, der die Änderung beschreibt.
  - **Gibt zurück** – Den Logger.
  - **Wirft**
    * `java.lang.NullPointerException` – Falls `entity` oder `key` nicht gesetzt sind.

  * `java.lang.IllegalArgumentException` – Falls `id` negativ ist oder kein zum Typ des Objekts passenden Schlüssel beschreibt.
  * `java.lang.IllegalStateException` – Falls zuvor die Initialisierung nicht abgeschlossen wurde, d.h. noch nicht der initiale Aufruf von `logStep` (in 2.1.3, Seite 13) erfolgt ist.

2.2 Schnittstelle LogWriter.Transaction

Die Schnittstelle Transaction.

Eine Transaktion gruppiert mehrere Änderungen während einer Runde an einem Objekt. Semantisch besteht dabei kein Unterschied zu mehrmaligen Aufruf von `notify` (in 2.1.3, Seite 13) erlaubt jedoch eine logische Gruppierung der Änderungen (Deltas) und die Möglichkeit das Objekt, welches an ein Spielobjekt gebunden ist, über Methoden weiter zu reichen.

Eine Transaktion wird erzeugt mittels `beginTransaction` (in 2.1.3, Seite 9) und wird in das Log mittels `commitTransaction` (in 2.1.3, Seite 10) übergeben. Beachten Sie hierbei, dass immer der zu letzt aktualisierte Wert geloggt wird.

2.2.1 Deklaration

public static interface LogWriter.Transaction

2.2.2 Methoden Überblick

set(LogWriter.Key, int) Fügt ein Schlüssel-Wert-Paar zu einer Transaktion hinzu.

2.2.3 Methoden

- **set**
  `LogWriter.Transaction` **`set(`**`LogWriter.Key` **`key, int value)`** `throws java.lang.NullPointerException, java.lang.IllegalArgumentException`
  - **Beschreibung**
    Fügt ein Schlüssel-Wert-Paar zu einer Transaktion hinzu.
  - **Parameter**
    * `key` – der Schlüssel, siehe `Key` (in 2.5, Seite 17).
    * `value` – der Wert welcher sich geändert hat.
  - **Gibt zurück** – die Transaktion
  - **Wirft**
    * `java.lang.NullPointerException` – sofern `key` nicht gesetzt ist.
    * `java.lang.IllegalArgumentException` – falls `key` nicht zulässig ist für den Objekttyp.

2.3 Die Klasse LogWriter.Cell

Diese Enumeration definiert den Typ einer Kachel. Eine Karte kann 3 unterschiedliche Typen von Kacheln enthalten. Alle Gegenstände befinden sich initial (sofern vorhanden) auf Wasserkacheln.

2.3.1 Deklaration

public static final class LogWriter.Cell
extends java.lang.Enum

2.3.2 Felder Überblick

ISLAND Eine Inselkachel.
SUPPLY Eine Vorratsinsel.
WATER Eine Wasserkachel.

2.3.3 Methoden Überblick

valueOf(String)
values()

2.3.4 Felder

- public static final LogWriter.Cell **WATER**
  - Eine Wasserkachel.
- public static final LogWriter.Cell **ISLAND**
  - Eine Inselkachel.
- public static final LogWriter.Cell **SUPPLY**
  - Eine Vorratsinsel.

2.3.5 Methoden

- **valueOf**
  `public static LogWriter.Cell` **valueOf**`(java.lang.String` **name**`)`
- **values**
  `public static LogWriter.Cell[]` **values**`()`

2.3.6 Vererbte Element von class Enum

`java.lang.Enum`
clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

2.4 Die Klasse LogWriter.Entity

Diese Enumeration definiert den Typ eines Objekts, welches zu loggen ist.

2.4.1 Deklaration

public static final class LogWriter.Entity
extends java.lang.Enum

2.4.2 Felder Überblick

BUOY Eine Boje.
SHIP Ein Piratenschiff.
TREASURE Ein Schatz.

2.4.3 Methoden Überblick

valueOf(String)
values()

2.4.4 Felder

- public static final LogWriter.Entity **BUOY**
  - Eine Boje.
- public static final LogWriter.Entity **SHIP**
  - Ein Piratenschiff.
- public static final LogWriter.Entity **TREASURE**
  - Ein Schatz.

2.4.5 Methoden

- **valueOf**
  `public static LogWriter.Entity valueOf(java.lang.String name)`
- **values**
  `public static LogWriter.Entity[] values()`

2.4.6 Vererbte Element von class Enum

`java.lang.Enum`
clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

2.5 Die Klasse LogWriter.Key

Diese Enumeration definiert die Schlüssel zum identifizieren der Werte welche zu Objekten im Log gespeichert werden können. Beachten Sie, dass nicht jeder Schlüssel bei jedem Objekttyp anwendbar ist.

`BUOY` (**in 2.4.4, Seite 16**)

- FLEET
- VALUE (Markierung der Boje)
- X_COORD
- Y_COORD

`SHIP` (**in 2.4.4, Seite 16**)

- DIRECTION
- FLEET
- MORAL
- PC
- RESTING
- VALUE (Wert der Schätze, welche auf dem Schiff gelagert sind)
- X_COORD
- Y_COORD

`TREASURE` (**in 2.4.4, Seite 16**)

- VALUE (Wert des Schatzes)
- X_COORD
- Y_COORD

2.5.1 Deklaration

public static final class LogWriter.Key
extends java.lang.Enum

2.5.2 Felder Überblick

DIRECTION Die Blickrichtung eines Objekts.
FLEET Die Flottenzugehörigkeit eines Objekts.
MORAL Die Moral der Mannschaft auf einem Piratenschiff.
PC Der Programcounter einer Flottenintelligenz.
RESTING Die Anzahl der Runden die ein Objekt im Spiel verweilen muss bis zu seinem nächsten Zug.
VALUE Der Wert eines Objekts.
X_COORD Die x-Koordinate eines Objekts.
Y_COORD Die y-Koordinate eines Objekts.

2.5.3 Methoden Überblick

valueOf(String)
values()

2.5.4 Felder

- public static final LogWriter.Key **DIRECTION**
  - Die Blickrichtung eines Objekts.
- public static final LogWriter.Key **FLEET**
  - Die Flottenzugehörigkeit eines Objekts. Flottenidentitäten beginnen bei 0 (= 'a').
- public static final LogWriter.Key **MORAL**
  - Die Moral der Mannschaft auf einem Piratenschiff.
- public static final LogWriter.Key **PC**
  - Der Programcounter einer Flottenintelligenz.
- public static final LogWriter.Key **RESTING**
  - Die Anzahl der Runden die ein Objekt im Spiel verweilen muss bis zu seinem nächsten Zug.
- public static final LogWriter.Key **VALUE**
  - Der Wert eines Objekts. Siehe Beschreibung von `Key` (in 2.5, Seite 17).
- public static final LogWriter.Key **X_COORD**
  - Die x-Koordinate eines Objekts.
- public static final LogWriter.Key **Y_COORD**
  - Die y-Koordinate eines Objekts.

2.5.5 Methoden

- **valueOf**
  `public static LogWriter.Key` **valueOf**`(java.lang.String` **name**`)`
- **values**
  `public static LogWriter.Key[]` **values**`()`

2.5.6 Vererbte Element von class Enum

`java.lang.Enum`

clone, compareTo, equals, finalize, getDeclaringClass, hashCode, name, ordinal, toString, valueOf