Testprotokoll

# Introduktion

Wir haben einen Zoo gemacht, denn wir anschliessend auch testen wollten. Wir haben bei den Tests vor allem darauf geachtet, dass wir die Initialisierung der Objekte mit den Konstruktoren richtig machen können. Dafür haben wir folgende Unit Tests.  
Da wir sehr viel mit initialisieren haben, haben wir sehr viele Mock-Methoden erstellt.

# Unit Tests

|  |  |
| --- | --- |
| Abschnitt | Inhalt |
| Test Name | addZoowaerterToList. |
| Anforderungen | Ein Käfigleiter soll mit der addZoowärter-Methode einen unterstellten Zoowärter in seine Zoowärterliste hinzufügen können. |
| Vorbedingungen | Ein Käfigleiter mit einer vorhandenen ZoowärterListe. |
| Ablauf | Ein Zoowärter wird der Liste hinzugefügt. |
| Erwartetes Resultat | Test Succes. |

|  |  |
| --- | --- |
| Abschnitt | Inhalt |
| Test Name | removeZoowaerterFromList. |
| Anforderungen | Ein Käfigleiter soll mit der removeZoowärter-Methode einen unterstellten Zoowärter von seiner Zoowärterliste entfernen können. |
| Vorbedingungen | Ein Käfigleiter mit einer Zoowärterliste, die mindestens 1 Zoowärter beinhaltet. |
| Ablauf | Ein Zoowärter wird von der Liste entfernt. |
| Erwartetes Resultat | Test Succes. |

|  |  |
| --- | --- |
| Abschnitt | Inhalt |
| Test Name | countAnzahlZooWärterCorrect. |
| Anforderungen | Die Methode getZoowärteranzahl() sollte implementiert sein. |
| Vorbedingungen | Ein Käfigleiter mit unterstellten Zoowärtern. |
| Ablauf | Ein Käfigleiter wird abgefragt, wie viele Zoowärter in seiner Liste sind. |
| Erwartetes Resultat | Test Succes. |

|  |  |
| --- | --- |
| Abschnitt | Inhalt |
| Test Name | countListTier |
| Anforderungen | Das Programm sollte erkennen wie viele Tiere sich in einem Käfig befinden. |
| Vorbedingungen | Ein Käfig mit einer Tierliste, die Tiere beinhaltet. |
| Ablauf | Es wird überprüft, ob der Käfig die richtige Anzahl Tiere zurück gibt. |
| Erwartetes Resultat | Test Succes. |

|  |  |
| --- | --- |
| Abschnitt | Inhalt |
| Test Name | checkKaefigInitialization |
| Anforderungen | Es ist wichtig zu wissen, dass ein Käfig richtig mit den Tieren initialisiert werden kann. |
| Vorbedingungen | Ein erstellter Käfig mit einigen Tieren. |
| Ablauf | Es wird überprüft, ob die mitgegeben Werte des Käfigs auch richtig initialisiert werden. |
| Erwartetes Resultat | Test Succes. |

|  |  |
| --- | --- |
| Abschnitt | Inhalt |
| Test Name | checkZoowaerterInitialization |
| Anforderungen | Es ist wichtig, dass ein Zoowärter eine richtige Initialisation bekommt. Somit kann man davon ausgehen, dass ein Zoowärter Objekt richtig erstellt wird. |
| Vorbedingungen | Ein mit Konstruktor initialisierter Zoowärter. |
| Ablauf | Der Zoowärter wird initialisiert und danach wird überprüft, ob er die Werte richtig bekommen hat. |
| Erwartetes Resultat | Test Succes. |

|  |  |
| --- | --- |
| Abschnitt | Inhalt |
| Test Name | negativCheckLöwenInitialization |
| Anforderungen | Bei diesem Test haben wir einen negativen Test gemacht und das Zauberwort «expected = AssertionError.class» benutzt. Wir wollten schauen, was passiert, wenn man etwas Falsches eingibt. |
| Vorbedingungen | Ein erstellter Löwe mit Fressverhalten. |
| Ablauf | Wir provozieren absichtlich einen Fehler, aber gleichen den wieder aus, da es genau das ist, was wir auch erwarten. |
| Erwartetes Resultat | Test Succes. |

# Testfazit

Bei diesem Projekt ist, das wichtigste, dass wir viele Mock Methoden in der Testklasse aufweisen können. Wir haben sehr viel mit Strategie gemacht und dies hat viele Abhängigkeiten, deswegen auch diese viele Mock-Methoden. Alle Tests sind so aufgebaut, dass sie passen und nicht failen. Bei vielen Tests haben wir die Konstruktoren getestet und geschaut, dass die Initialisierung richtig verläuft und keine Fehler aufweist. Zusätzlich haben wir noch einige Methoden getestet, damit wir keine Fehler beim ausführen des Programmes aufweisen und wir unseren Ausführungs-Fluss behalten.

